FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Self-Adjusting Mechanisms For Edge-Based IIoT In Variable Demands Environments

**Guilherme Fernandes Machado Rocha de Sousa**

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Gil Manuel Gonçalves

Co-Supervisor: Eliseu Moura Pereira

April 2, 2022

# Self-Adjusting Mechanisms For Edge-Based IIoT In Variable Demands Environments

**Guilherme Fernandes Machado Rocha de Sousa**

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

April 2, 2022

# Abstract

The evolution of technology directly correlates to the growth of the industrial world. From the industrial revolution to the present day, electronic device usage has increased, making not just industries grow but also bringing society closer. This evolution was especially marked in the computing area, where in thirty years, personal computers have gone from being a costly machine, mainly operated by computer scientists, to becoming one of the world's hottest and "cheapest" commodities, practically turning into the every day, indispensable utensil.

One of the concepts resulting from the combination of many fields of Computer Science is the concept of the Internet of Things. The Internet of Things (IoT) describes the network of physical objects-"things" that have the purpose of connecting and exchanging data with other devices and systems over the Internet, which allows for a modernization, centralization, and automation of whole processes, while adding overhead in terms of data storage and processing, that is usually done by using the resources available in Edge/Cloud Servers. Given its ever-increasing popularity, industries started adapting IoT techniques into businesses, creating the Industrial Internet of Things (IIoT). Considering how much data is processed in an industry, careful task offloading to the Edge/Cloud is necessary for IIoT.

But if many of the already attempted task offloading solutions managed to save resources by primarily applying Artificial Intelligence concepts, they often followed a single line of pre-determined thought. Another lacking point in research was the general absence of simultaneous integration of the Edge and Cloud levels, though this could be justified, since the cost of acquiring Cloud solutions can be prohibitive.

Therefore, a mechanism capable of performing this type of task offloading efficiently was developed, as an attempt to improve upon previous research's lacking points, that would communicate with an existing distributed system, named DINASORE. DINASORE allows for code execution of Function Block based pipelines, within a Cyber Physical Production System. In the end, three algorithms capable of showing task offloading optimization results were tested, and it was found that although they displayed acceptable levels of efficacy, they also displayed critical flaws in terms of accuracy. However, accuracy flaws were influenced by DINASORE's way of storing the monitored resources and monitored Function Block execution in text files and by the mechanism's attempt at establishing an association between this data. It was not possible to obtain a concrete and absolute understanding on the benefits of including a Cloud level with task offloading mechanisms, when compared to the negatives.

**Keywords**: Internet of Things, Industrial Internet of Things, Task Offloading, Edge Servers, Cloud Servers

# Resumo

A evolução da tecnologia está umbilicalmente ligada à evolução do mundo industrial. Desde a revolução industrial até à atualidade, que se verifica um aumento na presença de dispositivos eletrónicos, não só em termos industriais, como em termos sociais, pois aparelhos eletrónicos são utilizados nas mais comuns atividades do quotidiano. Esta evolução foi especialmente acentuada na área da computação, onde num espaço de trinta anos, os computadores pessoais, deixaram de ser uma máquina bastante dispendiosa e utilizada apenas por ciêntistas, ou informáticos, para se tornarem numa das mais baratas e mais utilizadas comodidades, praticamente indispensáveis no dia-a-dia.

Um dos conceitos, resultantes da combinação das várias áreas das Ciências da Computação, é a Internet of Things (Internet das Coisas). A Internet of Things (IoT) descreve a rede de objetos físicos que têm o propósito de conectar e trocar dados com outros aparelhos e sistemas através da Internet, o que permite uma modernização, centralização e automação de processos. Ao mesmo tempo, é gerada uma sobrecarga, no que respeita ao armazenamento e processamento de dados, realizado muitas vezes, ao utilizar os recursos disponibilizados em Servidores Edge/Cloud associados a soluções IoT. Dada a sua popularidade, as industrias começaram a adaptar técnicas IoT aos seus negócios, o que originou na Industrial Internet of Things (IIoT) (Internet das Coisas Industriais). Considerando a quantidade de dados que são necessários de processar e operar numa indústria, é necessário fazer um balanceamento cuidadoso de tarefas para a Edge/Cloud no decorrer das operações diárias executadas em ambiente industrial.

Mas se muitas das soluções de balanceamento de tarefas, apresentadas em investigações científicas realizadas pela comunidade científica, conseguem encontrar sucesso a poupar recursos, muitas das vezes seguem uma linha de pensamento já pre-determinada. Outro aspeto imporante que raramente foi estudado foi a integração conjunta dos níveis Edge e Cloud, o que pode ter sido influenciado pelo elevado custo de adquirir soluções Cloud.

Foi desenvolvido um mecanismo capaz de fazer este balanceamento de forma eficiente, como uma tentativa de melhorar os pontos fracos encontrados em investigações prévias, adequando-se a um sistema distribuído já existente, o DINASORE. Este sistema permite a execução de código de pipelines de Function Blocks em Sistemas de Produção Cyber-Físicos. No final, foram testados três algoritmos capazes de realizar balanceamento de tarefas, e os resultados demonstraram que embora houvesse algum sucesso do ponto de vista da eficiência, verificaram-se também falhas graves no que respeita à precisão, sendo que algumas dessas falhas foram influenciadas pelo modo como o mecanismo tenta estabelecer uma ligação entre os dados de consumo originados pelo DINASORE, tanto para dispositivos como Function Blocks. Não foi possível concluir de forma inequívoca, que a inclusão de uma solução Cloud com mecanismos de balanceamento de tarefas traga mais benefícios do que prejuízos.

**Keywords**: Internet das Coisas, Internet das Coisas Industriais, Balanceamento de Tarefas, Servidores Edge, Servidores Cloud

# Acknowledgements

I would like to thank Professor Gil Gonçalves for the opportunity presented, and Professor João Reis and Eliseu Pereira for all the feedback and guidance provided throughout the dissertation.

To all of the people that I have shared a classroom, field or locker room, I would thank for the many timeless memories created, that fueled the fire that aided in accomplishing all the goals I had set for myself at the beginning of my academic career. Namely, to my friends that came from Instituto Superior de Engenharia do Porto, with whom I shared many struggles, conquers and defeats, especially to my "brother in arms", Joel Coelho for his friendship and work ethic, which inspired and helped me completing my Masters Degree.

To all my friends and loved ones, whose undying and unwavering support and love was a critical foundation in this short life that I have lived so far.

To my family for all the understanding and opportunities created, especially to my sister, mother and father, who provided nothing but love, affection and care since the day I was born, and who have given me all the necessary conditions to succeed in life as well as the weapons needed to face all of its adversities.

You have all inspired me to become a better person and to never give up. To all of you my eternal gratitude and my undying love.

Guilherme Sousa

*“Ten decisions shape your life,*
*You’ll be aware of five about.”*

Julian Casablancas

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| 4DIAC | 4 Distributed Industrial Automation and Control |
| ACO | Ant Colony Optimization |
| AI | Artificial Intelligence |
| CS | Computer Science |
| CPS | Cyber Physical Systems |
| CSV | Comma-Separated Values |
| CVR | Conditional Value at Risk |
| CPPS | Cyber Physical Production Systems |
| DIGI2 | Digital and Intelligent Industry Lab |
| DINASORE | Dynamic INtelligent Architecture for Software and MOdular REconfiguration |
| FEUP | Faculty of Engineering of the University of Porto |
| FB | Function Block |
| GB | Gigabytes |
| GLBS | Gupta-Livne Bargaining Solution |
| ICS | Industrial Control Systems |
| IEC | International Eletrotechnical Commission |
| IoT | Internet of Things |
| IIoT | Industrial Internet of Things |
| ISO | International Organization for Standardization |
| MDP | Markov Decision Process |
| MEC | Mobile Edge Computing |
| MQTT | Message Queuing Telemetry Transport |
| OASIS | Organization for the Advancement of Structured Information Standards |
| OPC-UA | Open Platform Communications-Unified Architecture |
| PSO | Particle Swarm Optimization |
| QoS | Quality of Service |
| RAM | Random Access Memory |
| SCADA | Supervisory Control and Data Acquisition |
| SOM | Self-Organizing Map |
| SYSTEC | Research Center for Systems and Technologies |
| TABS | Tempered Aspirations Bargaining Solution |
| XML | Extensible Markup Language |

# Chapter 1

# Introduction

In this chapter, the research is presented and briefly discussed to understand its context, objectives, questions, and planning. The document's structure is also presented.

## 1.1 Context

The relationship between technology and the industrial world may not be umbilical since the production of goods has been done since Men populate our Planet, who used nothing but their hands to produce items and equipment for activities vital to the prosperity of the human species. But the invention and continuous advancement of technology provided the industrial world with a perfect partner for symbiosis, as technology was ultimately an essential spark that would justify and shape the industrial revolution of the 19th Century. And if at first, the revolution began with simple machinery and materials, such as coal and iron, which, at the time, were enough to change the world, the truth is, the tremendous changes verified in the technological world, namely in the computing area, have expanded industries in ways beyond imaginable.

The rapid and exponential change in technology benefited not only industries but also society as a whole since some electronic devices are considered an essential good. From mobile phones to computers, the ubiquitous presence of these technological marvels shape the everyday life of a vast majority of the world's population and change the way it is possible to communicate, consume, collaborate and behave. Specifically, the area of Computer Science (CS) has experienced an immense amount of evolution since the invention of the first digital computer. Concepts such as Artificial Intelligence (AI), Concurrent, Parallel and Distributed Computing, Databases and Data Mining, Security and Cryptography, Computer Architecture and Organization, have been defined and studied by some of the brightest minds in CS, and from being interesting theoretical ideas, became understood and regularly used concepts, not only by other scientists but also Software Engineers and developers.

One of the most recent and exciting concepts, is the concept of Internet of Things (IoT). The IoT represents the network of physical objects embedded with electronic frequency identification, software, sensors, actuators, and smart objects that converge with the Internet to accumulate and share data with other systems and devices [5]. Fields like wireless sensor networks, control and embedded systems, and others enable the Internet of Things. Still, in terms of the consumer market, IoT technology is synonymous with products relating to the concept of the "smart home" like devices and appliances that support one or multiple common ecosystems, which are controllable via devices, such as smartphones and smart speakers [27].

Despite its original market target, other sectors of society have found value in applying and integrating IoT into organizations to modernize, centralize, and automate whole processes and businesses. By applying IoT techniques to industries, the Industrial Internet of Things was then created.

Industrial Internet of Things or IIoT corresponds to interconnected sensors, instruments, and other devices networked together with computers' industrial applications, which allows for data collection, exchange, and analysis, which facilitates efficiency and productivity improvements, as well as other economic benefits [8]. Concepts such as Cloud Computing, Edge Computing, Big Data, Artificial Intelligence, Machine Learning, and, of course, Internet of Things come together to allow process controls' refinement and optimization, in IIoT [33]. But even if IIoT has plenty of value for industries, it is necessary to accommodate resources and plan solutions individually depending on needs, often defined by how demanding the environment in which each industry is inserted.

Therefore, this research aims to provide and document steps necessary to develop a load-balancing mechanism for Edge-based networks that will allow to optimize its performance and guarantee the system can fulfill the requirements imposed by variable demand.

## 1.2 Research Focus

The research was done under a proposition by the Digital and Intelligent Industry (DIGI2) Lab, which is part of the Research Center for Systems and Technologies (SYSTEC), Faculty of Engineering of the University of Porto (FEUP). DIGI2 develops Research and Development activities in advanced control concepts and systems, tools, and technologies for a broad area of industrial applications. Their mission is scientific and technical research, development, advanced education and training, dissemination, and sustainable technology transfer. [1]

To understand the focus of this research, an example is provided. One of the main concepts in the industrial world, is a production line. Cambridge Dictionary defines a production line as a line of machines and workers in a factory that a product moves along while it is being built or produced [2]. As an example, a shoe production line can be used. In a production line, there could be separate lanes for products. Supposing, in the context of a shoe factory, that the first lane would responsible for the feet of the shoes, the second, for the body of the shoes, and the last one for the laces. By using IIoT techniques and technologies, it should be possible to perform actions such as product identification, product picking, product sorting, and others, that generate data that often needs to be collected, processed, and analyzed. But to store data, Edge and Cloud solutions are preferred since they provide a lot of value to companies by mainly allowing to reduce costs and maintenance efforts, provide agility in terms of existing or new infrastructure and versatility. If the production line only processed about one-hundred shoes, it wouldn't offer much problem in terms of managing the Edge and Cloud's resources, therefore it would be necessary to choose only one of them to always keep storing data resulting from said processing. But if the production line had to process one-hundred thousand shoes, the resources needed to process all the data would increase, and in this case, only using one of the solutions would not suffice. Therefore, it would be necessary to balance how and in which solution data would be processed for optimizing the resources that are spent, which can be done using an offloading mechanism.

Offloading, in computing, corresponds to the task of sending computation-intensive application components to a remote server [4]. Offloading to an external platform over a network can aid in breaking through hardware limitations, such as limited computational power, storage, and energy, and consequently, make resources more available for computation operations. The research's focus was then to design and implement a mechanism capable of accurately and efficiently perform task allocation suggestions, according to resources available, in an environment where large amounts of data need to be processed in real-time and stored in Edge or Cloud solutions accordingly.

## 1.3 Research Relevance

Though previous researches were done about task offloading in Edge and Cloud solutions, many of these types of papers usually targeted the consumer market definition of IoT, so they were applied to the concept of smart-home instead of industry, though research was found that provided a solution, specifically for IIoT task offloading by, for example, using Reinforcement Learning techniques. While many of the findings will be presented and further discussed in the next chapter of the thesis, it is possible to say that this research would provide inspiration for an attempted solution for task offloading optimization suggestions. The solution would consist of an overview architecture and at least one algorithm that defined how the offloading itself should be performed.

## 1.4 Research Objectives

As said in section 1.1, the research aimed to provide and document steps necessary, to come up with a mechanism for Edge-based networks that would provide suggestions on how to optimize the performance and to guarantee a system could fulfil the requirements imposed by variable demand. To achieve this, the architecture was designed, and algorithms would be defined to understand and measure what type of strategy was the most efficient at giving suggestions of task offloading. More specifically, the research's objectives were:

- To define the most important variables in the environment when designing a task offloading mechanism.
- To design an abstract architecture for said mechanism, capable of fitting a vast majority of solutions without having to modify the existing infrastructure too much.
- To implement at least one algorithm capable of accurately and efficiently understanding how the tasks should be balanced.
- To analyze how efficient and accurate the architecture and resulting algorithm(s) were in the suggestion of load-balancing of tasks' solutions.

## 1.5 Research Questions

To guarantee the objectives defined were accomplished, research questions were also defined. These questions would set the guidelines to focus on and define the means necessary for achieving the objectives:

- What parameters defined how important a variable in the environment was ?
- How abstract can a load-balancing mechanism's architecture be to achieve a satisfactory portability parameter and efficient task execution ?
- How should the algorithms be defined to achieve efficient task execution ?
- How to measure the efficiency and accuracy of the proposed solutions ?

Therefore the main research question was:

> "How to define an abstract architecture and develop, at least, an algorithm capable of efficiently and accurately offloading tasks associated with demanding industrial environments embedded within the Industrial IoT?"

## 1.6 Thesis Structure

The thesis is structured as follows:

- **Chapter 1** introduces the research by presenting its context, motivation, objectives, focus and relevance.
- **Chapter 2** presents the state of the art of existing task offloading solutions for Edge/Cloud solutions in IoT/IIoT and some concepts related to Reinforcement Learning.
- **Chapter 3** describes the work problem formulation.
- **Chapter 4** presents the mechanism's architecture, the implementation phase and all the steps taken towards the development of the offloading algorithms.

- **Chapter 5** describes the process of testing the mechanism developed and discusses the results obtained during the testing phase.
- **Chapter 6** presents the mechanisms' limitations as well as mentions future work that could be performed to overcome them.
- **Chapter 7** presents the conclusions obtained from the research.

# Chapter 2

# Literary Review

In this chapter, the State of the Art of task offloading solutions will be presented, and further context will be given regarding the Internet of Things and the Industrial Internet of Things. To better understand how this chapter's structure was defined, it is necessary to know how the research was structured, planned, and executed. Given that task offloading is a concept associated with the Internet of Things as a whole, it was necessary to plan the research to accommodate this factor. Though the research didn't follow a traditional systematic review process, it was still possible to follow a plan that didn't differ much from it. To make things easier, the researches performed always included a set of keywords, which included task offloading, Internet of Things, Industrial Internet of Things, offloading mechanism. To filter out the relevant information from the irrelevant one, papers were read to understand their quality/relevance, which were parameterized accordingly to methodology, technologies used, experiments/simulations executed for testing mechanisms, and results presented. Given the differences in environment between IoT/IIoT systems, different sections will present the existing work for two different versions of IoT, the consumer-oriented one and the industrial version. A section related to the theory and concepts behind Reinforcement Learning will be presented to provide more clearance to some of the terms and definitions that impacted developmental choices.

## 2.1 Internet of Things/Industrial Internet of Things

This section presents more context about the Internet of Things and the Industrial Internet of Things. Starting with the Internet of Things, as stated in chapter 1, section 1.1, it describes the network of physical objects that converge with the Internet to accumulate and share data with other devices and systems, and according to Kumar et Al., is an emerging paradigm that is capable of facilitating lives [9]. Specifically, IoT offers developments in technology capable of merging in people's everyday lives, such as the concept of Smart Home Systems. These systems are composed of appliances consisting of Internet-based devices, automation systems for homes, and reliable energy management systems that communicate with each other to make the management of homes much easier by automating processes such as house cleaning and managing home security and in-home media entertainment systems. Another area of society where the IoT has an enormous number of benefits is the healthcare area with Smart Health Sensing Systems. These health systems incorporate indoor and outdoor equipment and devices to monitor/check health issues, fitness levels, or burned calories by a patient. More importantly, they can be used to monitor critical health conditions. IoT also has relevance in transportation since, when integrated with vehicles, IoT systems can improve traffic awareness and efficiency. The main advantage of IoT systems is that devices and equipment are cost-effective when it comes to development and product purchase, managing to find their way into many people's hands. But all through the explosion of the technology, research is still done on the many facets of IoT, making it more and more relevant and utilized in the global market. Hardware such as sensors, RFID chips, and barcode readers mix with networking concepts such as 3G/4G, Wi-Fi, Bluetooth to create the basis for IoT architecture, composed of 5 layers, perception layer, network layer, middleware layer, application layer, business layer. However, it is necessary to account for scalability, modularity, interoperability, and openness to fulfil multi-system integration requirements with functionalities related to management, storage and big data analytics/studies and storage, user-friendly applications, and cross-domain interactions. But even if IoT has a lot of advantages, it also presents several risks and issues such as:

- **Security/privacy issues** can arise when authentication and authorization flows are missing or incomplete, the software is insecure, or there is inefficient/missing transport layer encryption.
- **Interoperability/standard issues** can be found due to the heterogeneity of technologies and solutions used to develop IoT systems.
- **Ethics, law, and regulatory rights** should be attended to since data security, privacy protection, trust, and safety data usability can represent a challenge in the development of IoT systems.
- **Scalability, availability, and reliability** given that IoT systems need to account for a large number of devices, which display different parameters and attributes while understanding that availability and reliability are, more often than not, crucial in the good functioning of these systems.
- **Quality of Service (QoS)** that serves as a measure to understand not only how efficient IoT devices, architectures, and systems are but also how they perform and how they evaluate in terms of quality.

Some of the main fields where the Internet of Things has relevance and brings benefits include:

- **Emerging economy, environmental and health-care**, where IoT systems have importance in accomplishing social, health, economic and environmental goals.

- **Smart city, transport, and vehicles** as discussed, some of the main areas where IoT first showed its capabilities, allow for more efficient household and traffic tasks to be performed.
- **Agriculture and industry automation** when IoT can improve processes through automation and can work to resolve environmental concerns. It is precisely in the industrial variant of IoT that the research focuses on.

In [26] Boyes et Al. presented an analysis framework on the Industrial Internet of Things. IIoT, or as it is also known, Industry 4.0, is considered a collective term for concepts and technologies of value chain organization. The Industrial Internet of Things is related to concepts like Cyber Physical Systems (CPS), Supervisory Control and Data Acquisition (SCADA), Industrial Control Systems (ICS), and Industrial Internet, which are a part of IIoT ecosystems. Cyber Physical Systems comprise a set of digital and physical components capable of interacting with each other through centralized or distributed architectures that provide functions that influence the real world through physical processes. Namely, they are used to aid in controlling the physical processes while displaying large degrees of autonomy. Industrial Control Systems are different control systems with associated instruments such as systems, devices, networks, and controls used in industrial processes. Supervisory Control and Data Acquisition systems allow for an operator in a strategic physical location to perform tasks such as monitoring alarms, managing valves, set point changes on distant process controllers, and gather measurement information. Finally, Industrial Internet is concept-oriented to the industrial world, as it combines two components, the first is the connection of industrial machine sensors and actuators to local processing and to the Internet, and the second is the forward connection to other important industrial networks, each capable of generating value. These systems allow for the collection and treatment of data so that it can be analyzed and possibly used to influence new value-generating services. The Industrial Internet is also known as a short-hand for the industrial applications of IoT, or simply, as the Industrial Internet of Things. The IIoT uses smart objects within CPS's, in order to achieve industrial goals and to connect industrial components such as power grids, engines, and sensors to cloud over networks. But IIoT systems rely on more than just CPS's, they are also associated with generic information technologies and optional Cloud or Edge computing platforms, capable of enabling collection and analysis of data, autonomous and intelligent access, exchange of process within the industrial environment, to optimize overall production value, making it possible to reduce energy consumption, reduce labor costs, or even improve the delivery of services and products. Boyes et Al. also provided "compartmentalization" of IIoT devices depending on the following criteria:

- **Industry sector** since all different sectors in the industrial world require different devices to function properly.
- **Device location** is considered from many perspectives since a device location can cause it to be more exposed to risks from a security perspective.
- **Device connectivity** given that it is necessary to identify essential features of connectivity between an IIoT system and devices that are constituents of that system.

- **Device characteristics** to understand how devices function, namely, what type of importance they have on the system, and what kind of functions they perform, or how they should be managed.
- **Device technology** needs to be attended to because, usually, technological features constraint device design or the capacity to address vulnerabilities when devices have been deployed.
- **User type** is essential to define since it tracks the identification of who is using the device and what the device is interacting with.

Finally, the authors identified some of the issues/needs in terms of research within the IIoT theme that need to be taken into consideration in the future:

- **Limited research on safety and security of IIoT devices** which could potentially be a point of harm in industrial systems. Therefore additional work needs to be performed to prevent and minimize the threat to personnel, assets, and the environment.
- **Mapping the IIoT ecosystem and threat landscape** to understand which security threats can expose and harm IIoT ecosystems and associated threat landscape mapping to those ecosystems.
- **Limited research on Operational Technology and Information and Communication Technologies convergence** given that merging these two areas requires well-defined, scalable standards while also guaranteeing that these standards are secure and don't expose any possible vulnerability.
- **Providing solutions to legacy system issues** given that many industries still have legacy systems in place to perform every day it is necessary to account for the processes and security definitions in place when installing IIoT devices on top of an operational architecture.

## 2.2 Concepts

Following the guidance of the many papers presented in the following sections, one of the algorithms developed for the mechanism was a Q-Learning algorithm. It is then necessary to further clarify theoretical concepts and definitions associated with Reinforcement Learning. Concepts such as Markov Decision Process, state, action, environment will be presented, since their understanding is critical to better comprehend how that particular algorithm was built.

### 2.2.1 Reinforcement Learning

Reinforcement Learning teaches an agent a specific behavior by interacting in a trial and error manner with a dynamic environment. Considered to be a class of problems, Reinforcement Learning solutions may follow two different strategies. The first one entails that a behavior space must be queried until one appropriate behavior for the environment is found. In contrast, the second strategy uses statistics and dynamic programming techniques to estimate a utility value obtained from performing given actions in all the worlds' states. The Q-Learning algorithm was developed based on the second strategy, which justifies presenting the concept of the Reinforcement Learning model. As shown in Figure 2.1 a model is constituted by many different parts. One of those parts is the agent who interacts with an environment, defined as the agent's world. This interaction is done via perceptions and actions, occurring after receiving data. Data as input ($i$) and states ($s$) will be the main influence in the action ($a$) chosen by the agent to be its output. The action chosen affects the state of the environment. This effect is quantified via a reinforcement signal, also referred to as reward ($r$, while $R$ represents the reward function), and will influence the agent's long-run behavior ($B$) via a trial and error execution. The agent can perceive the different environment states due to an input function ($I$). Simultaneously, by mapping states to actions capable of maximizing the reinforcement signal, the agent can accomplish its true goal, which is to find a policy that manages to do that maximization efficiently. [24]

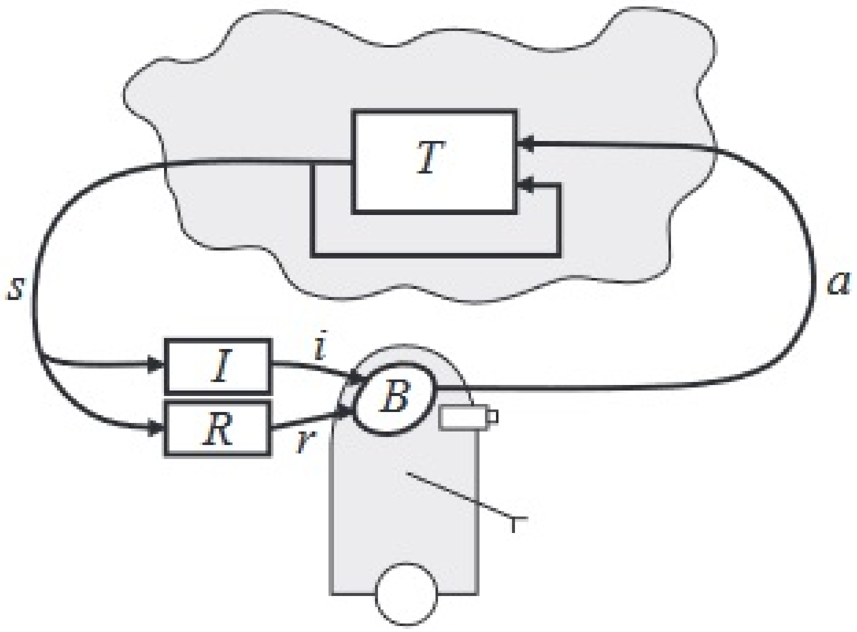


Figure 2.1: Reinforcement Learning Model (taken from [24])

All of the concepts presented above are related to the concept of Markov Decision Process (MDP), which defines a sequential behavior decision problem, describing the agent. As explained, this agent has a value function associated with it that returns the sum of all the expected rewards. In turn, this value function is associated with the Bellman equation. The environment defined for an MDP is probabilistic, meaning that the state transition and reward obtained will be randomized after an action is performed. [23]

More concisely, the following concepts are fundamental in MDPs.

#### 2.2.1.1 State

Can be defined as a set of situational observations performed by the agent.

#### 2.2.1.2 Action

Can be defined as a set of actions an agent can perform at a given state, which are the same for every other state.

#### 2.2.1.3 State Transition Probability Matrix

Corresponds to the representation of agent movement in the sense that after taking a certain action, an agent goes from one state to a new state with a defined probability. The following function represents it:

$$P_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a) \tag{2.1}$$

#### 2.2.1.4 Reward

The reward is given to the agent by the environment, so the agent can learn the different compensation expected values obtained from performing different actions. The following equation can be used to represent the reward:

$$R_{ss'}^{a} = E[R_{t+1} | S_t = s, A_t = a] \tag{2.2}$$

#### 2.2.1.5 Discount Factor

Is considered to be a depreciation of the reward so that as time passes, the value of the reward the agent obtains diminishes. This way, it provides a solution to possible overcompensation problems.

#### 2.2.1.6 Policy

The policy is used to aid the agent in selecting the action to take at a certain state. The objective of Reinforcement Learning is to keep on learning the best possible policy to obtain an optimal policy. The policy can be formulated as follows:

$$\pi(s|a) = P(A_t = a | S_t = s) \tag{2.3}$$

#### 2.2.1.7 Value function

The value function is considered to be the sum of the rewards expected to be received in a state after following a defined policy. It is the value function that indicates which policy is the optimal policy. The following equation represents the value function:

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma * V_\pi(S_{t+1}|S_t = s)]. \tag{2.4}$$

Where $E_\pi$ corresponds to expected value, $\gamma$ represents the discount factor and $R_{t+1}$ is the reward value to be awarded afterwards.

#### 2.2.1.8 Action Value function

If the previous formula represented the state value function, in the sense that it returns the reward sum value for when a state is given, the action-value function takes not only the state into consideration but also accounts for the selected action. Therefore the agent uses the action-value function, or Q-function, to choose the action appropriately, and the Q-function can be represented as:

$$Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma * q_\pi(S_{t+1}, A_{t+1}|S_t = s, A_{t+1} = a)] \tag{2.5}$$

The relationship between the value function and the function in equation 2.5 can also be mathematically detailed in an expression where the policy is added to the value of the Q-Function for every action:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) Q_\pi(s,a) \tag{2.6}$$

#### 2.2.1.9 Bellman Equations

The Bellman equations are the mathematical representation of the relationship between the value functions of two states, the one the agent is currently situated in and the state the agent will be situated in after performing an action. There are two different equations, the Bellman Expectation Equation and the Bellman Optimality Equation. If the former is responsible for finding the optimal policy after receiving the greatest expectation for all the policies, the latter is the one that obtains all of those expectations. The Bellman Expectation Equation can be defined as described in equation 2.7, while the Bellman Optimality Equation is presented in equation 2.8.

$$v_{\pi'}(s) = \sum_{a \in A} \pi(a|s)(R_{t+1} + \gamma \sum_{s' \in S} P^a_{ss'} v_{\pi'}(s)) \tag{2.7}$$

$$v_*(s) = \max_a E_\pi[Rt + 1 + \gamma v_*(S_{t+1}|S_t = s)] \tag{2.8}$$

#### 2.2.1.10 Q-Learning

Considered to be single agent, the basic Q-Learning algorithm is an off-policy method that separates the learning policy from the acting policy. Not suited for multi-agent environments, this algorithm uses many of the concepts presented in the previous subsections to develop a Q-function named the Q-value. The Q-value equation is presented in 2.9 where $\alpha$ is a value between 0 and

1, and it is considered to be a learning rate, while $R$ is the reward and $\gamma$ the discount factor. Q-Learning is an iterative process, where at each step, the Q-value is updated for every state by using the Q-value equation. Before this process starts, a Q-table is instantiated, and in it, all of the rewards are stored. An agent can select actions based on policies, and the transition itself is made possible using the state transition probability matrix. Q-Learning is a repetitive process because the overall Q-value needs to converge to a specific value to where the Q-table can then be used to solve the problem at hand. By combining Monte Carlo and dynamic programming methods, one of Q-Learning's main advantages is the simplicity and effectiveness with which it provides results, although its once per action update and its memory requirements limit its effectiveness for multi-agent environments.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max Q(s',a') - Q(s,a)] \tag{2.9}$$

With the information obtained from [24] and [23], it was possible to present many theoretical concepts that will enable a better understanding of one of the algorithms developed for the research.

## 2.3 IoT Based Task Offloading

Though the scope of the thesis was related to the Industrial Internet of Things, the majority of previously performed work is mainly related to the consumer version of the Internet of Things. Even when understanding the significant differences verified in the demand of an industrial environment, it is still possible to affirm when compared to the need of the home setting, that the type of investigation performed in previous experiences and corresponding findings could still be used as inspiration and guideline for the research due to the similarity of the problem they were trying to solve. Namely, the papers proposed new ways of performing task offloading while being cautious and aware of spent resources. Resources such as energy, time, storage, availability were studied and defined as basis for many of the proposed solutions that attempted to minimize said resources while maximizing performance.

To understand the terminology used in the papers, the concepts of Fog, Cloud, and Edge computing are presented. In a very simplistic explanation, Cloud computing aims at empowering service providers by facilitating efficient resource management in data centers, which allows for reducing costs and increasing scalability, though latency can be problematic. Edge computing corresponds to a distributed computing model that collects data at the edge of the network in order to process it in real-time. The benefits of Edge computing are reduced bandwidth use, which saves money and avoids bottlenecks, increased security by encrypting at the source while addressing the drawbacks of the Cloud by reducing latency. Finally, Fog computing provides "closer" computation, data storage, and application services when compared to Cloud computing while essentially being the bridging element between the Cloud with the Edge [32]. Figure 2.2 shows an architecture of interaction between the three concepts.

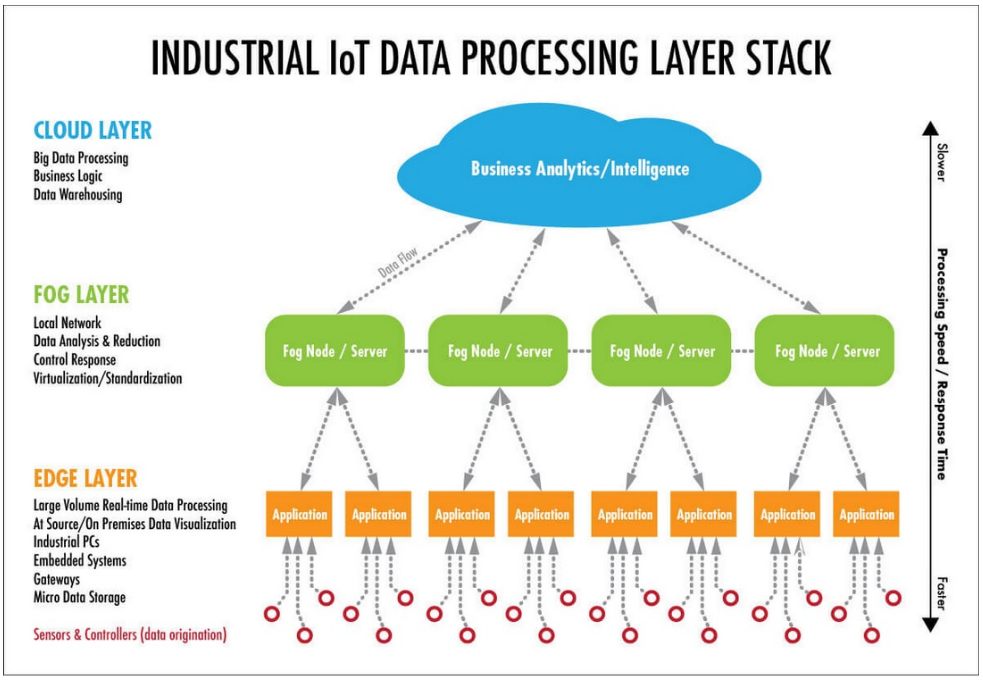

INDUSTRIAL IoT DATA PROCESSING LAYER STACK
CLOUD LAYER
Big Data Processing
Business Logic
Data Warehousing
FOG LAYER
Local Network
Data Analysis & Reduction
Control Response
Virtualization/Standardization
EDGE LAYER
Large Volume Real-time Data Processing
At Source/On Premises Data Visualization
Industrial PCs
Embedded Systems
Gateways
Micro Data Storage
Sensors & Controllers (data origination)
Business Analytics/Intelligence
Data Flow
Fog Node / Server
Application
Slower
Processing Speed / Response Time
Faster

Figure 2.2: Interaction Architecture Between the Fog, Cloud and Edge (Taken from [7])

In [3] Aazam et al. grouped the different task offloading mechanisms proposed up to late 2018 in research where one of its main contributions was the presentation of various criteria used for task offloading in IoT. The criteria for offloading in the majority of the existing literature are:

- **Excessive computation** where offloading occurs when computational resources needed far exceed the capability of the native device by executing tasks in resource-rich device(s).
- **Latency requirements** dictate that offloading is needed since distance can affect delay-sensitive applications. In this case, a node close to the user's proximity will offload the task and provide the required services with minimum delays, often sending parts or the entire content to be cached at Edge devices or Fog servers.
- **Load balancing** happens when one server has reached its capacity of executing tasks, provoking that additional tasks are distributed among other servers to balance their processing.
- **Permanent or long-term storage** since offloading is often performed to manage storage better. For example, it is not feasible to perform services that require long-term storage on small devices, which justifies the usage of Cloud servers for heavy data processing.
- **Data management and organization** serves to show the findings resulting from the activities performed in the research.
- **Privacy and security** where depending on how important the data or tasks are, offloading can occur to guarantee privacy and security.

- **Accessibility** can influence offloading to a Cloud-like node when the native device from where the task or data was created is not accessible from everywhere.
- **Affordability, feasibility, and maintenance** is the final criteria for offloading, as the cost and feasibility of maintaining high-end servers are incredibly high, which can be reduced by "outsourcing" data storage and processing to other services such as Clouds, whose providers take responsibility for the maintenance of said services.

As for algorithms and models for solving the task offloading problem, it was possible to identify different experiments and results in various researches. In [14] Fang et Al. proposed an efficient strategy for computational offload with Mobile Edge Computing (MEC) by using fine-grained tasks that had dependencies and modeling user-generated mobile applications as a directed acyclic graph. This was done to make the parallel processing of tasks possible while combining it with a Multi-population Co-evolutionary Elite Genetic Algorithm (MCE-GA) to solve resource allocation and task scheduling problems, therefore reducing the overhead of the generated applications. In this model, computation offloading was performed either locally or in Edge servers, and since tasks were dependant on one another, they needed to be executed orderly and had to wait for their turn at being executed. The time that a task had to wait before executing was defined as ready time, which tied with the problem's formulation, and considered as a fundamental concept in determining the total delay and energy consumption for executing tasks locally or in a given Edge server node. The total delay of applications generated by a user corresponded to the sum of the total delay of a given task when executed locally plus the total delay of the same task when executed at an Edge server node. The total energy consumption was calculated similarly, the only difference being the sum of the power consumption between the local and Edge node environments. The MCE-Genetic Algorithm would use these concepts in its fitness equation as it was possible to observe in equation 2.10, where the Beta variable, a value between zero and one, represented the trade-off parameter of delay and energy since some tasks required low latency while others required low energy consumption. After the initialization and selection of the population, two intersection points were randomly set in the individual. Then, partial gene exchange was carried out to preserve its diversity and better solve offloading decisions. The MCE-GA was executed alongside a Random Selection algorithm and a Greedy algorithm to verify its effectiveness, and it was proven via simulation results that the proposed algorithm found reasonable allocation of resources and reduced the overhead of the whole system. It also proved itself superior to the other algorithms in terms of reducing overhead in user equipment.

$$Fitness = \beta * TotalConsumptionPower + (1-\beta) * TotalDelayTime \tag{2.10}$$

Similarly, in [22] two meta-heuristic algorithms, inspired by nature, named Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), were proposed as two different scheduling algorithms to balance the load of IoT tasks over Fog nodes under response time and communication costs considerations. Here the total communication cost when a task was offloaded consisted of the sum of the cost needed to offload the same task from the device to the Fog node and the cost of offloading it from the Fog node to the Cloud node. The service response time was defined as the sum of the average service time on the Fog nodes, the Fog-Cloud communication cost, and the IoT-Fog communication cost. Both algorithms tried to implement task offloading that complied with Quality of Service constraints, such as the response time, the service time, and the amount of load on the Fog nodes. A Round-Robin algorithm was also used to measure how effective the proposed mechanisms were. The evaluations showed that the ACO algorithms bettered IoT applications' response times compared to the PSO and Round-Robin algorithms while being more effective in load balancing the Fog nodes.

Perhaps one of the most fascinating and complex task offloading mechanisms was proposed in [16] by Gao et Al., consisting of a Predictive offloading method for resource allocation in multi-layered Fog computing systems, where within each layer one Edge node (Edge Fog node) and a Central Fog node resided. The distinguishing aspect of this proposal was that in addition to the local processing and offloading queues, there existed two other queues, the prediction and arrival ones, wherein the former, untreated workload predicted to arrive within a specific time-slot was stored, and the latter stored workload that arrived at a given Edge-Fog node, ready to be processed either in the local queue or the offload queue. The predicted queue was updated at the end of pre-service, and the look-ahead window pushed one slot ahead when each time slot terminated. Finally, the integrate queue had an input consisting of the predicted workload, while the output consisted of workloads moved to the offloading queue and the local processing queue. The Central Fog Node also had queues for arrival, offloading and local processing of tasks though they functioned similarly to queues in the Edge Fog node. The problem was formed as a minimization problem, where there was an attempt to minimize CPU frequency of the Edge Fog nodes, the overall power consumption, and the amount of workload to be either locally processed or offloaded. To solve it, Lyapunov optimization techniques were used to simplify the problem into a subset of problems over time slots. The predictive algorithm allowed for each Fog node to decide the amount of workload to go into the local processing and offloading queues respectively. To achieve optimal results, each Edge Fog node's integrate queue size was compared with its local processing queue size and offloading queue size. Concretely if there were significantly more workload in the integrate queue when compared to the local queue, tasks would be offloaded as much as possible to be processed locally. Similarly, if there were significantly more workload in the offloading queue, when compared to the integrate queue, the tasks would be offloaded as much as possible to be processed in the corresponding queue. Notably, if the backlog size of a given Edge Fog node integrate queue was larger than both its local and offloading queue, the workload would be transmitted one by one, being sent either to the local queue or the offloading queue. Nevertheless, the amount of workload never surpassed each of the queue's sizes respectively. For simulations, the paper's authors verified if a given Edge Fog node's integrate queue backlog size was greater than the size of the backlog of the local and offloading queues. If this condition were true, the tasks would be transmitted to the local queue for processing until the local queue's capacity was filled. The remaining tasks would be transmitted to the offloading queue until its capacity was also filled. The process ended when the integrate queue was empty. With such definitions, the Predictive algorithm could achieve a tunable power-latency tradeoff while at the same time effectively reducing this latency, even when only mild values were used in the prediction window for future information and, more interestingly, the effectiveness of the solution wasn't very much affected even when there were prediction errors.

Even more outstanding from a technical standpoint was described in [25], where Sungwook Kim proposed the TABS (Tempered Aspirations Bargaining Solution) and GLBS (Gupta-Livne Bargaining Solution) algorithms. These algorithms consisted of mobile device/Cloud cooperative bargaining games where all the components in a multi-layered system interacted with each other through negotiations of their respective conflicts, resulting in freeing resources through a fair and efficient process. In this experiment, devices offloaded their requests to either Fog nodes or one Cloud node. In the offloading process, the device would send its applications' tasks while at the same time partitioning the computation amount into three different parts to be executed in the local device, the Fog node, and the Cloud node, respectively. Both algorithms had resulted from previous works, done on bargaining methods for negotiation, where TABS could be characterized by following axioms such as Symmetry, Scale Invariance, *r*-Restricted S-Monotonicity, Irrelevance of Trivial Reference Points, S-Continuity and Weak Pareto-Optimality, while GLBS, though still following the Weak Pareto-Optimality, Symmetry, and Scale Invariance axioms, also followed the Relevant Domain, *d*S-Monotonicity, and Limited*d*-Sensitivity axioms. TABS and GLBS were used to implement a time-sensitive and computation-oriented application offloading algorithm, respectively. The selection of which algorithm to execute was made dynamically depending on the type of application generated by the device. If an application task was computation-intensive but delay-tolerant, the bargaining solution would be measured by taking as a reference point the starting point, since the task could be executed without offloading services, justifying the use of the GLBS algorithm. On the other hand, in cases where tasks had restrictive time limits, the bargaining solution was measured by taking as a reference point the disagreement point between constituents of the system, which justified using the TABS algorithm. In the end, simulations were used to test the scheme proposed, which consisted of six steps, the first was the determination via said simulations of the control parameters and system factors, followed by the equal generation of different types of application tasks. The next step was conditional, if the application task generated in a given period was computationally intensive, GLBS would be selected to process the offloaded task, guaranteeing an effective distribution of the application tasks between the device, Fog node, and Cloud node, but if the application generated was time-sensitive TABS was used in the processing of the offloading service guaranteeing a dynamic distribution of application tasks between the device, the Fog node and the Cloud node. The computational resources, namely the load of these system constituents, were then monitored online and in real-time. Finally, the system itself checked for changes in the network and re-triggered the process if a new application task was generated, as seen in figure 2.3. It was possible to conclude that the proposed mechanism successfully managed and distributed resources in a system to improve task execution while guaranteeing the adapted bargaining methods were effectively fair in offloading the tasks to devices, Fog nodes, and Cloud nodes.

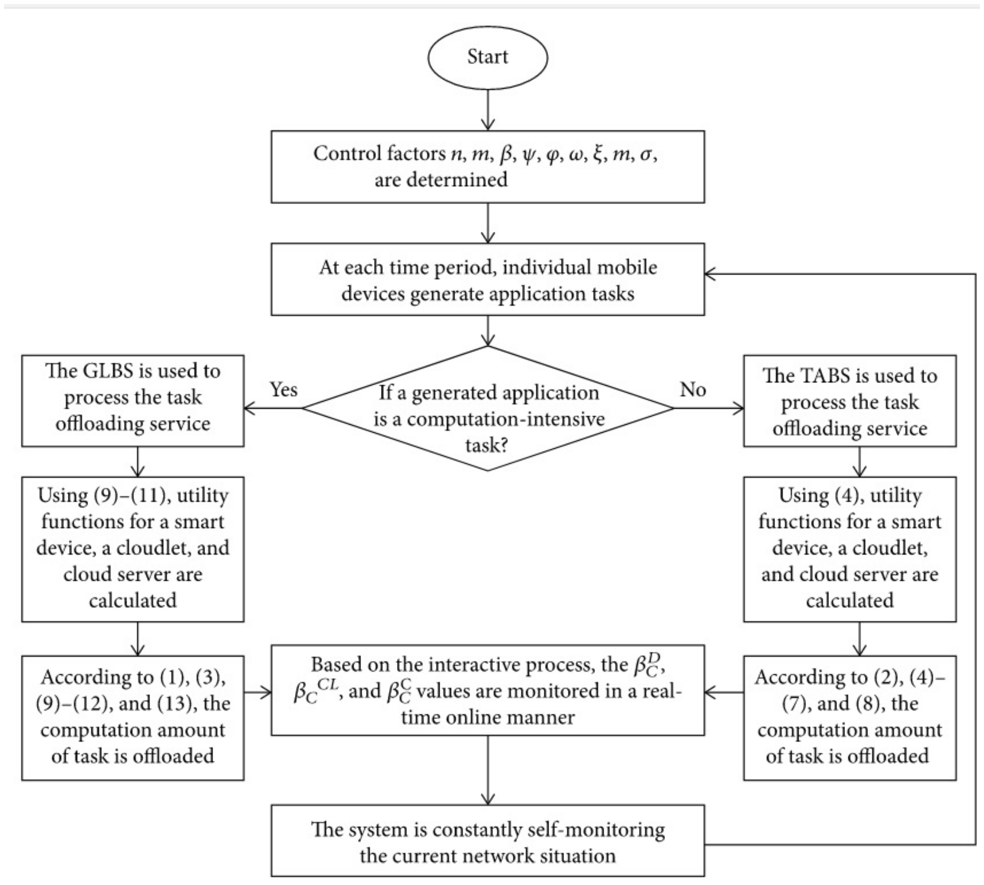

Start
Control factors $n, m, \beta, \psi, \varphi, \omega, \xi, m, \sigma$, are determined
At each time period, individual mobile devices generate application tasks
If a generated application is a computation-intensive task?
Yes
No
The GLBS is used to process the task offloading service
The TABS is used to process the task offloading service
Using (9)–(11), utility functions for a smart device, a cloudlet, and cloud server are calculated
Using (4), utility functions for a smart device, a cloudlet, and cloud server are calculated
According to (1), (3), (9)–(12), and (13), the computation amount of task is offloaded
Based on the interactive process, the $\beta_C^D$, $\beta_C{}^{CL}$, and $\beta_C^C$ values are monitored in a real-time online manner
According to (2), (4)–(7), and (8), the computation amount of task is offloaded
The system is constantly self-monitoring the current network situation

Figure 2.3: Bargaining based scheme for task offloading (Taken from [25])

But even if Artificial Intelligence concepts heavily influenced the majority of these proposed mechanisms, other possibilities were related to more functional approaches for developing task offloading mechanisms, though not as automated as those presented before. One such case, resulting from the research by Li et Al., in [28], was proposed in the form of an energy-aware task offloading framework. But even if its title only referred to energetic resources, the requirements of the framework focused on other resources, namely, transmission delay, by deploying several Edge servers at the Edge of IoT, to improve network capacity, offer real-time response for IoT devices, and computing resource allocation, wherewith the aid of the Edge server, IoT devices offloaded their tasks to the Edge server to reduce the load on the IoT devices. To achieve an optimal task offloading strategy, it was necessary to minimize the transmission delay and energy consumption, as reducing energy consumption resulted in improving the number of completed tasks during their Time To Live. According to the authors, the optimal task offloading strategy could be found by comparing the completion time between the IoT device and the Edge server. If the time to finalize task m were less than the task's completion time at the Edge server, task m would be considered invalid. If the remaining time of task m were less than the completion time at the IoT device but higher than the completion time at the Edge server, task m would be executed by the Edge server. If task m had a higher remaining time than the completion time of the IoT device, the costs of performing task m for the IoT device and Edge server would be compared. If the transmission power consumption cost, and the overall power consumption cost, of performing task m at the IoT device were higher than power costs for performing task m at the Edge server, the task would be performed by the Edge server. Otherwise, task m would be performed by the IoT device. When it came to results, a scenario where there was one Edge server and 100 IoT devices was considered to compare the proposed solution to the conventional ones. It was possible to verify that the optimal scheme obtained a higher task completion ratio than that of the conventional ones and also improved on the number of completed tasks and lowered energy consumption as it is possible to observe in figure 2.4. One of the most interesting aspects of the research was the fact that the authors brought up concepts that should be very important in the future of task offloading solutions, like Content Caching-Enabled Edge computing, an important way to improve the efficiency of content distribution in IoT, Mobility-Aware Cooperative Edge Caching, which will provide flexible resource utilization while facilitating content caching and delivery over Edge-based solutions and finally, Privacy-Aware Applications for Edge computing, since security and privacy are considered to be one of the fatal flaws of IoT infrastructures.

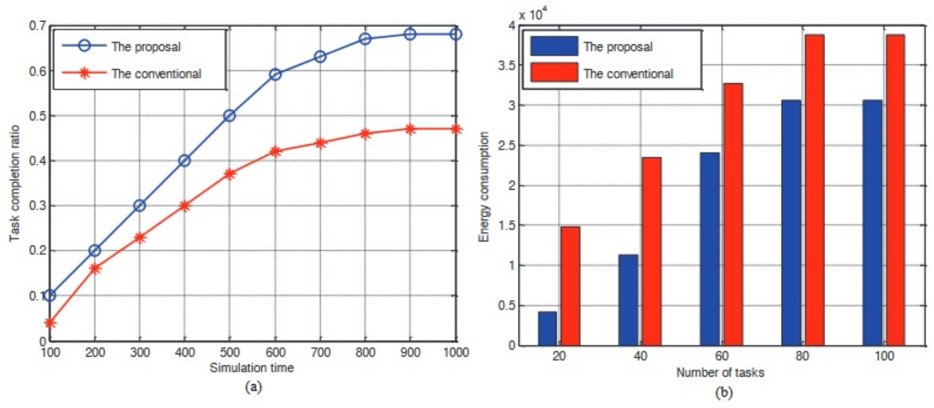


Figure 2.4: Results obtained by comparing the optimized scheme to the conventional scheme (Taken from [28])

In [15] Flores et Al. proposed AutoScaler as an attempt to provide offloading as a service. The main advantage of the work was that AutoScaler was designed and implemented, motivated by Service-oriented architectures, such as Amazon AWS AutoScale that, essentially, served as an interface between tasks that needed to be offloaded and the available resources within Cloud nodes by scheduling tasks among those nodes, allowing for a study of the capacity of this offloading architecture to handle different levels of workload. The authors also defined models for implementations of offloading, the first, Homogeneous model where the device's and server's Runtime Environment were the same, and the code of the task was present in both of them, which could be helpful when there was no Internet connection. The Heterogeneous model, where the Runtime Environment was different for both the device and server, which meant they also had different implementations of tasks, therefore in this case, only input parameters were transferred in the offloading communication. Finally, the Neutral model where the Runtime Environment did not influence task offloading since the code was exclusively stored in the server, ready to be called by a device, though this could only happen in the presence of Internet connectivity. AutoScaler was categorized as Homogeneous, and to better understand the models, figure 2.5 is presented.

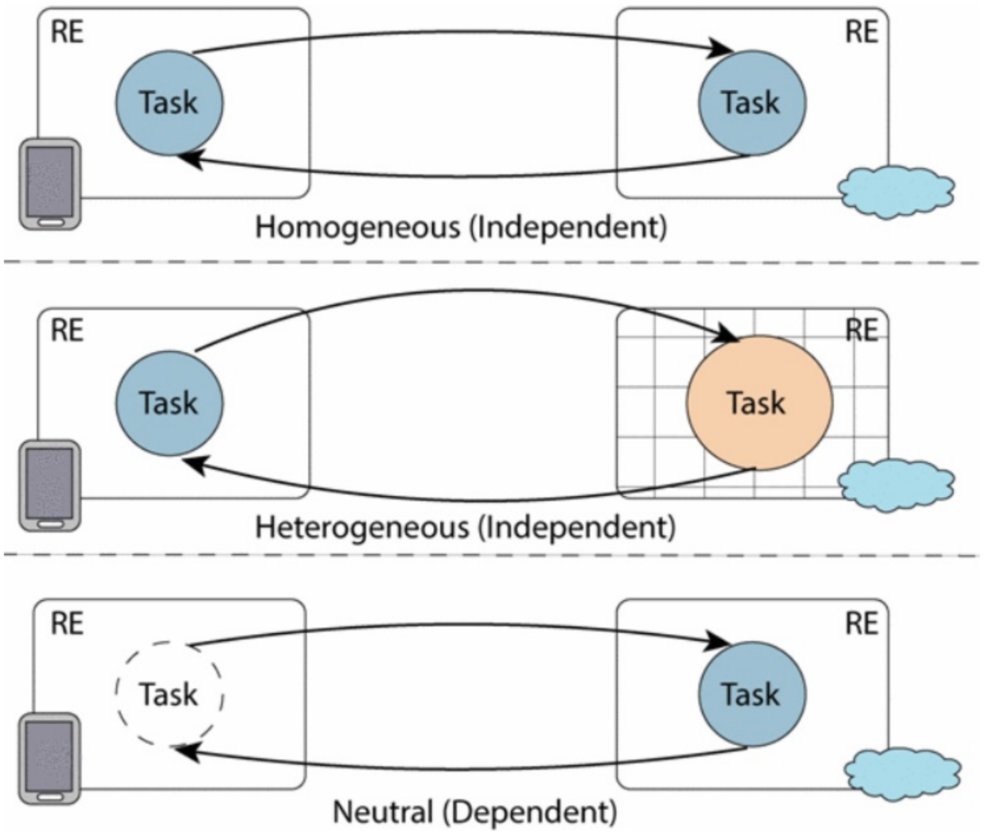


Figure 2.5: Types of offloading models (Taken from [15])

The system consisted of three parts, the first, the back-end, where every node was a customized Dalvik Virtual Machine of Android to execute Dalvik bytecode, whose main benefit was the ability to activate multiple instances of the same applications or running multiple applications concurrently. Secondly, the front-end, or better yet, AutoScaler itself, distributed the load between the servers available through a round-robin scheduler. It also contained a file that kept the information about APK file localization and the available ports of servers for offloading, and this file was updated when a server was either removed or added. The allocation of servers happened when the response time of the workload being offloaded passed a predefined threshold. Finally, the simulator was used to generate different loads of device offloading to the Cloud. The overall system architecture is presented in figure 2.6.

The experiences done to test AutoScaler had in consideration performance and scalability metrics. In terms of metrics, it was possible to verify that even as the load increased, the time to distribute requests between servers for AutoScaler was approximately one-hundred and fifty mil-

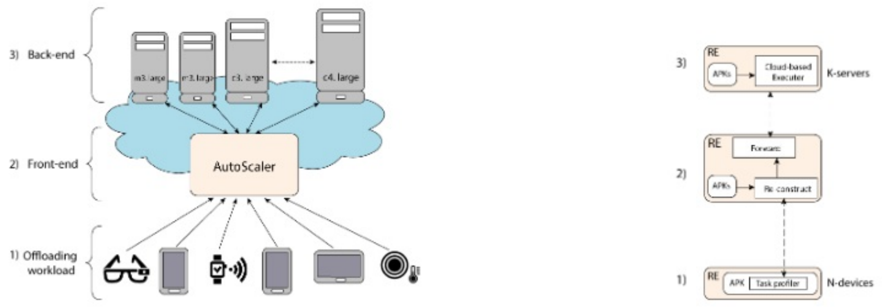


Figure 2.6: System Architecture (Taken from [15])

liseconds. Furthermore, it was verifiable that back-end nodes would start consuming more CPU resources to process the tasks, limiting the number of tasks they could handle simultaneously. However, this consumption could be reduced, by executing code in a parallel manner, since requests in the system were processed all at once to achieve shorter response times, and therefore, scheduled by AutoScaler. As for performance, AutoScaler introduced extra-time to the time necessary to respond to an offloading request in about one-hundred and fifty milliseconds, but this value is possible to be reduced by processing tasks in the Cloud and by meticulously managing the tradeoff between the price for utilization and computational resources of the server selected for offloading.

## 2.4 IIoT Based Task Offloading

Though there weren't as many studies and papers specifically done for IIoT systems in relation to those that exist for IoT systems, the existing literature for IIoT thematics was very interesting and diverse in what they propose and achieve. In [17], Hao et Al. used concepts from the financial area to propose an optimal offloading policy. The basis of the work was risk management theory, using the Conditional Value at Risk (CVR) concept to understand how the delay could be captured, and the upper bound of CVR was obtained from analyzing the states of the local queue and the queue of the Edge servers. To solve the problem itself, the authors considered the average delay performance and the risk and treated task offloading as a weighted sum of both these parameters. Therefore the proposed mechanism aimed to minimize the maximum value of the mean sum of the risk among all devices that would be involved in the offloading. It was considered an NP-hard non-convex mixed integer nonlinear problem, where tasks generated by devices could only be offloaded to one Edge server. For an Edge server, the number of devices that it served couldn't exceed its number of CPU cores. At the same time, the sum of the computation frequency allocated to the devices couldn't exceed its overall computation frequency. The most interesting part was that the proposed mechanism was formulated and solved more in a mathematical way than via software, using mathematical proof and linear programming, so while it didn't generate a lot of interest, from a technical standpoint, it was still a good possible solution, since the strategy was able to control the risk of delay jittering existing and at the same time making sure the performance delay stayed at average levels.

Moving on to more technical propositions, Chen et Al. used an accelerated gradient algorithm to try and achieve energy minimization and delay guaranteed offloading solution in [11]. To accomplish this, a model for minimizing energy was developed, considering the energy consumption of Fog nodes, including the local computation devices. The model was supposed to help lower the energy consumption as much as possible when the tasks' completion time was kept at an expected level. The authors defined the total energy consumption of a node, but to understand the formula, it is necessary to understand a variable they named as $a_i$, that corresponded to the ratio between the size of the tasks that were offloaded and the total tasks' size. As it is possible to observe in equation 2.11 the total energy consumption consisted of the sum of the energy consumption of executing a computation task locally, times one minus the $a_i$ variable, with the value of total energy consumption of transmitting a task to a Cloud server times the $a_i$ variable, with the energy consumption of a Fog node when it was at idle state.

$$e_i = e_i^l * (1 - a_i) + (e_i^c * a_i) + \triangle * e_i \tag{2.11}$$

$$t_{all} = t_i^l * (1 - a_i) + (t_i^c * a_i) \tag{2.12}$$

To calculate the total time needed for a task to complete, the formula, presented in equation 2.12 is more straightforward, it consisted of the sum of the time of task completion in local nodes with the time it took to transmit a task to be offloaded, and the time it took to execute an offloaded task at the Cloud node. Besides energetic concerns, the authors also considered channel bandwidth and computation resources in their proposed optimization offloading solution. It should be noted that while the objective was to minimize the consumption of energy in all Fog nodes when processing tasks, the total energy consumption had to be less than a defined desired energy consumption, and the completion time of tasks could not exceed a chosen desired delay. To understand the optimal values the mechanism should display to guarantee these restrictions, the authors used an accelerated gradient algorithm to solve the problem. When it came to results, the algorithm improved the energetic consumption on local computing nodes. However, it made the energetic cost of transmitting the task to the Cloud worst while improving completion time and guaranteeing the delay constraint was complied with. The value of $a_i$ converged to the optimal value as more iterations of the algorithm were executed, and it was possible to achieve minimum energy consumption after running one hundred iterations.

In [12], Dao et Al. proposed an online pattern task identifier mechanism to perform task offloading by identifying the most used task patterns in IIoT and trained a Self-Organizing Map (SOM) that, within defined dimension boundaries, represented the tasks' features themselves. This approach was thought out since no existing work could improve the performance of Edge computing systems when tasks needed to be executed online, or in simpler terms, when, on arrival, the tasks at Edge nodes needed to go to the queue before any scheduling decision, was made. Yet again, experiments made by the authors were done on top of a multi-layer Edge computing system, where its devices would generate tasks that needed to be offloaded, but the main differentiation factor was the use of a Self-Organizing Map. A Self-Organization Map is essentially an artificial Neural Network that utilizes Unsupervised Learning to obtain, as the name says, a map. A SOM has a set number of neurons that reflect the map's dimension, and it functions in two modes, training mode, where a map based on a set of tasks was created, and mapping mode, where created tasks that arrived were classified. Figure 2.7 shows how a SOM is represented.

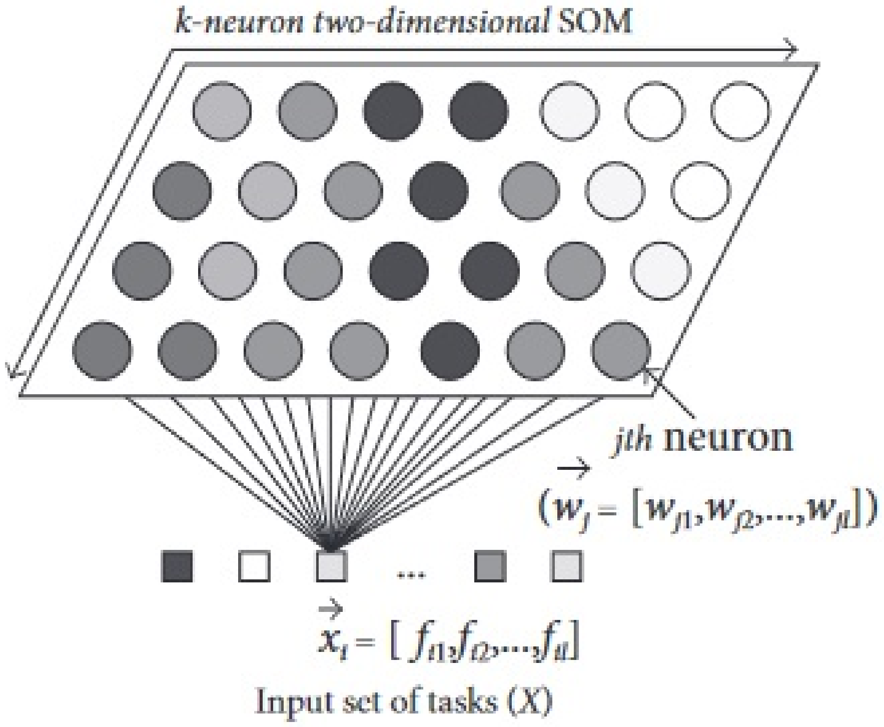

k-neuron two-dimensional SOM
jth neuron
$(\vec{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jl}])$
...
$\vec{x}_t = [f_{t1}, f_{t2}, \ldots, f_{tl}]$
Input set of tasks (X)

Figure 2.7: Self-Organizing Map (Taken from [12])

To facilitate the identification of typical task execution in IIoT, a given task was considered by the authors as being a four-dimensional feature vector given by:

$$\vec{x}_i = [u_i, c_i, r_i, \tau_i] \tag{2.13}$$

The parameters were relative average processing complexity ($c_i$), relative response size ($r_i$), relative execution deadline ($\tau_i$), and relative task size ($u_i$). With these settings, the SOM was trained by gathering all offloaded IIoT tasks in a distinctive period, considered as a cycle counting from the moment the task arrived after being offloaded. It was after the training that the neurons in the SOM would have a weighted value, a reflection of the historical occurrences of tasks, or better yet, a reflection of the values present in the vector defined in equation 2.13. To understand the optimal task assignment that should take place at each time slot, it is necessary to understand total latency was calculated. This parameter consisted of the sum of the value of uploading latency, with the value of queuing latency, task processing latency, and response latency. To solve the problem, it was necessary to minimize the values of latency while guaranteeing that a neuron in the SOM could only be assigned to one Edge computing node, in the system, by resorting to the Hungarian method and applying it to the latency matrix of all possible task assignment from a set of neurons to an Edge computing node set. To work in the online mode, as soon as a task arrived at the Edge computing system, a match to the SOM was performed to seek the optimal task assignment. To test the mechanism proposed, twenty Edge computing equipment with different and multiple CPU frequencies were used in a system with three-hundred IIoT devices in total, and three types of tasks were defined according to the type of data traffic involved in them:

- **Environmental sensor data** where during each fixed period, the environmental conditions data were sent to applications in the IIoT central system.
- **Video surveillance data** that consisted of data resulting from monitoring cameras' live feed sent to surveillance applications, stored and analyzed in the Edge framework.
- **Production control data** corresponded to data generated during specific production processes and was generated by the machine.

Alongside the proposed solution, the evaluation was done by running iterations of the same tasks using the Hungarian method, but for the offline mode and also for an online Greedy task assignment algorithm. When it came to results, it was possible to conclude that the latency of task processing was reduced when the proposed mechanism was used for offloading compared to the other methods evaluated. For time-consumption and average time decision making, the mechanism proposed performed better than the offline Hungarian method but scored worst time than the online greedy task assignment algorithm, given this method performs fewer calculations than the online mode Hungarian method, even though the difference between the two methods didn't affect task processing latency proving the capability for the stability of the method. The solution also reduced the number of execution errors present in short deadline tasks, meaning tasks that had to be executed in a minimal amount of time, while also reducing the buffering during the transmission phase.

Similar to the literature previously presented, Hong et Al., proposed a computation offloading mechanism for an IIoT system, based on a concept taken from the domain of Economics/Game Theory, which is Nash Equilibrium, in [19]. The primary differentiation in this work was that the solution tried an approach to achieve a multi-hop and cross-layer task offloading mechanism capable of cooperation and of assuring Quality of Service. The majority of existing offloading approaches assumed that all IIoT devices had a wireless Internet connection, enabling them to communicate with Cloud and Edge servers. Still, often in the real world, the Internet connection could be faulty, making it difficult to connect to the Cloud/Edge nodes. Therefore in such cases, devices collaborated to transmit and bring data closer to the nodes where the offloading should be performed, resulting in a multi-hop computation offloading mechanism. To make this mechanism cooperative across the different layers of the system, the IIoT devices, the Cloud, and the Edge, two algorithms that would try to achieve a Nash Equilibrium were implemented. The model defined by the authors considered that all tasks that were intensive for computational resources were of the same type and utilized the same CPU frequency. To allow for the mechanism to function under the game concept, every device needed to set a strategy, or every device needed to choose the path for data to follow. As it is possible to see in both equations 2.14 and 2.15, a strategy consisted of a tuple with the variable $a_i$ and the variable $p_i$. The first variable represented an integer that would indicate the choice for computation of the task. In contrast, the second would store the sequence that the task would follow from the device where it generated to the target, which could be a Cloud or Edge node. For local computing, the cost of executing tasks on a local device took into consideration the time it took to compute the task itself, plus the energy spent on the processing, combining it with a weighted factor given to time and energetic parameters involved in the strategy chosen by the device. As for Edge computation, the fact that devices might have needed to transmit tasks between each other before reaching the target influenced the total cost of executing tasks at this level, while the weight factors that existed in local computing still affected Edge costs. Finally, for the Cloud costs, the calculations included the step where tasks would be offloaded from the Edge server to the Cloud server since Cloud servers usually provide much more resources than their Edge counterparts. Therefore the cost of executing a task at a Cloud node was influenced by the time it took for it to compute, by the time it took for a device to offload the task to the Cloud and also by the time it took for a task to be offloaded from the Edge node to the Cloud node. The weight settings were once again the same. An important parameter that influenced the solution was the relay cost, given that devices needed to act as relays to guarantee that tasks could arrive at their processing destination in case of Internet failures.

$$s_i = (a_i, p_i) \tag{2.14}$$

$$a_i = \begin{cases} 0, & \text{if i computed locally} \\ 1, & \text{if i computed by offloading at the edge} \\ 2, & \text{if i computed by offloading to the cloud} \end{cases} \tag{2.15}$$

For the purpose of game formulation, devices needed to display a certain degree of independence, to choose the offloading strategy considered to be the best, which meant that each device would choose the strategy that would allow them to reduce the cost of operations. The game was formulated by considering devices as players who had a set of valid strategies and associated costs. Though devices chose a strategy that would better suit their desires, they could switch strategies when confronted with one that could prove more efficient, in what was called an improvement step, that could be categorized in the following ways:

- **Local / Cloud** if the cost of executing a task at a local device was less than the cost of executing it at the Cloud node and $a_i$ was two, it could improve from Cloud to the local environment. Otherwise, if $a_i$ was zero and the cost situation was inverse, it could improve from local to the Cloud.

- **Local / Edge** if the cost of executing a task at a local device was less than the cost of executing it at the Edge node and $a_i$ was one, it could improve from Edge to the local environment. Otherwise, if $a_i$ was zero and the cost situation was inverse, it could improve from local to Edge.
- **Cloud / Edge** if the cost of executing a task at a Cloud node was less than the cost of executing it at the Edge node and $a_i$ was one, it could improve from Edge to the Cloud computing. Otherwise, if $a_i$ was two and the cost situation was inverse, it could improve from Cloud to Edge.
- **Path Change** if $a_i$ was either one or two, the device could improve by changing its parent and using a different offloading path.

With this information at hand, the first algorithm was thought out, named free–bound improve algorithm, that as the name says, used to its advantage the fact that devices could act as relays to introduce the concept of being bound. Simply, if a device was acting as a relay, it would be considered to be bound, otherwise, it would be considered free, and this state for a device could change during the execution of the game. The multi-hop cooperative offloading mechanism was designed for allowing that a set of free devices could reach a Nash Equilibrium by improving their path choice through the usage of the free-bound algorithm, as can be seen in figure 2.8. The strategy changes that could occur in the cases presented in the referred figure were done when:

- **Case 1** the cost of executing a task at an Edge node was less than the cost of executing the same task at the local device, causing the free device to change the strategy from local computing to Edge computing. The strategy could also follow the opposite way, meaning it could change from Edge computing to local computing.
- **Case 2** the cost of executing a task at a Cloud node was less than the cost of executing the same task at the local device, causing the free device to change the strategy from local computing to Cloud computing. The strategy could also follow the opposite way, meaning it could change from Cloud computing to local computing.
- **Case 3** the cost of executing a task at a Cloud node was less than the cost of executing the same task at an Edge node, causing the free device to change the strategy from Edge computing to Cloud computing. The strategy could also follow the opposite way, meaning it could change from Cloud computing to Edge computing.
- **Case 4** in either Edge or Cloud computing, a free device could choose to offload via a different path by changing its parent to a different device to keep transmitting data to be offloaded.

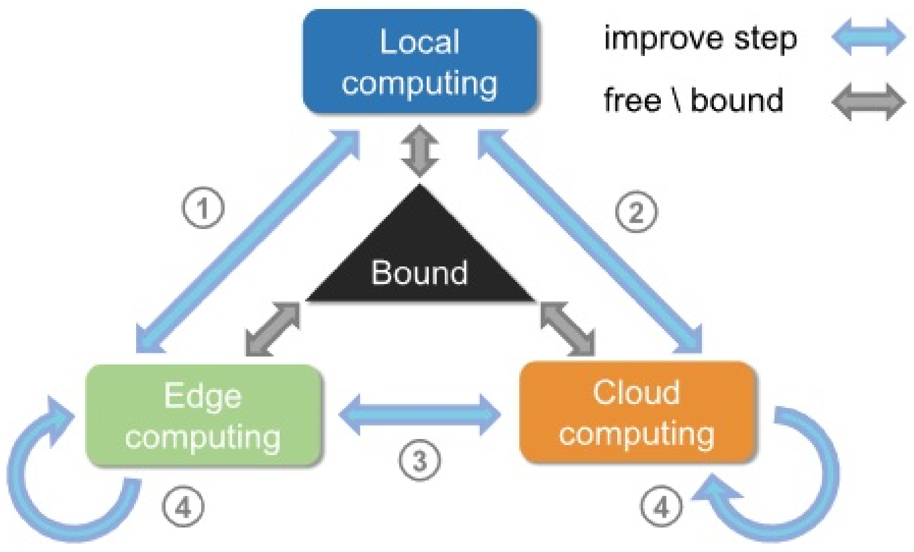


Figure 2.8: Free Bound Mechanism (Taken from [19])

Unfortunately, this algorithm alone wasn't enough to achieve an efficient Nash Equilibrium, and to help on this, a distributed algorithm that was QoS-aware was designed. Though a good theoretical approach, the free-bound algorithm, didn't account for three factors:

1. The fact that a device needed to understand how each path was filled, to know how congested it was, before making a decision.
2. The fact that, due to privacy and security concerns, devices might have held some information when communication with other pieces of hardware was required.
3. The fact that in the free-bound algorithm, to find a Nash Equilibrium, it needed to be guaranteed that when improvement steps occurred, they occurred only once and only for one device.

So, to guarantee the multi-hop cooperative method functioned properly, messages were sent alongside the data on device communication, which simplified understanding of what kind of task occurs during communication. The types of message data are:

- **Information Message (IM)** for each device that transmitted data, it would only transmit information about its number of relays and its transmission rate.
- **Strategy Message (SM)** when a device had received many Information Messages, this type of message was sent to base stations in the IIoT system and informed them of their intention to change strategy.
- **Allow Message (AM)** when a base station received a Strategy Message, a negotiation would allow the improvement to be made. A message of this type was sent to one device and only one time.

- **Begin Message (BM)** was a type of message that kept being sent to devices until a Nash Equilibrium was achieved.
- **Update Message (UM)** was a type of message sent to confirm a successful change of strategy. Sent by an IIoT device to all other devices present in the offloading path ($p_i$).

The distributed algorithm, named QoS-aware distributed algorithm, was divided into six phases. In the first phase, BM messages were broadcasted to devices in the network, a process repeated until all devices understood a game had to continue. After transmitting IM messages, phase two started to provide help to devices trying to choose a strategy. Phase three would be where devices effectively chose their strategy and transmitted that information to the base station devices. Phase 4 was the heart of the algorithm because, in phase 4, two different possibilities entered the table, either the game was terminated, meaning a Nash Equilibrium had been achieved since base stations received no SM messages. Nonetheless, if an SM message was sent to just one base station, then an AM message would be sent to a device to be chosen. This device selection was made, by introducing algorithmic variants to the QoS-aware algorithm, the first, named *FCFI*, in which a base station that received an SM message would authorize the device that sent the message to improve its strategy, and the second, named *ACTI* where the base station would only decide when after a certain time it had received every SM message and had evaluated them allowing the best improvement step to be chosen. Finally, phase 5 served for devices to update or retain its strategy, which set the stage for phase 6, where devices that weren't being used as relays were set as free devices, and those that needed to help others by relaying their messages would be set as bound devices. Figure 2.9 shows a high-grained flow-chart of how the mechanism used the messages to accomplish its objectives effectively.

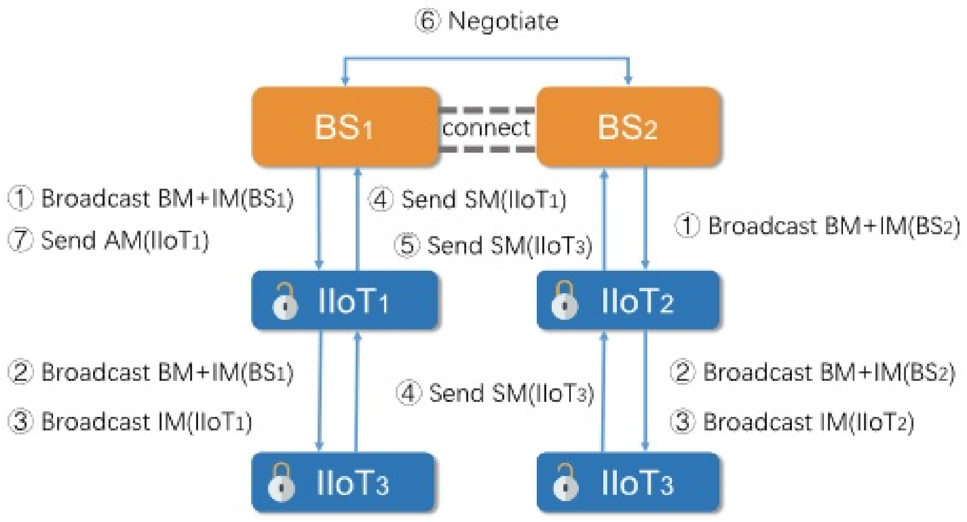


Figure 2.9: Flow Chart Messages in the Distributed Algorithm(Taken from [19])

In experiments, the proposed mechanism was tested alongside a Greedy algorithm and Simulated Annealing algorithms in a system where one-hundred and eighty devices IIoT and ten base stations were connected via the Internet. The results were varied, and many different parameters were evaluated, for example, for performance, the proposed mechanism proved far more efficient than the Greedy and Simulated Annealing ones, but it was slightly better for the ACTI variant than for the FCFI one, therefore being incredibly well suited and stable, when dealing with computationally intensive tasks. One interesting conclusion was that the more IIoT devices present in the process, the more effective the proposed multi-hop algorithm was, and while the process didnt't offer the absolute best strategy, it provided a result that was very near that value.

In [20] Hossain et Al. proposed a reinforcement learning task offloading scheme in one of the most relevant paper related to Industrial Internet of Things offloading solutions. The end devices were considered agents that decided whether the tasks were to be computationally offloaded to Edge devices belonging to an IIoT system. Specifically, the authors used a Q-Learning approach instead of the traditionally used Markov Decision Process approach to achieve optimal time-constraint values and evade one of the problems in the MDP approach, which is the fact that it is used for single-user task scheduling. With the Q-Learning method, it was possible to perform multi-user task scheduling while trying to better end-to-end delay and availability in the system when faced with an increase in the number of devices present and the correspondingly increasing number of actions that each device could perform. When it came to the system model, seen in figure 2.10 the authors considered an IIoT network equipped with end devices and edge nodes, similar to other models, with the exception that the Edge nodes had gateways from which it was possible to access them, to, once again, improve on metrics such as energy, time, computational capacity and latency. It was also defined that tasks, whether executed locally or on an Edge node, would spend the same amount of computing resources needed for execution, that for all end devices the offloading delay should not pass a defined value, which consisted a constraint for optimizing the scheme, while in this case tasks could only be offloaded directly. The novelty in this work was that for the decision-making process, a heuristic was defined based on parameters such as bandwidth, data size, energy consumption, trust, input/output data size, delay sensitivity. The problem was treated as a cost minimization problem to reduce the system's latency and power consumption.

For local computing, the delay of end devices included only the delay of offloading a certain task to the CPU and could be calculated as dividing the time required for the task to execute in the respective devices by the devices' total computational capacity, since individually, computational capacity was different between devices. The local energy consumption was obtained by multiplying energy consumption for every Cloud cycle to complete a task by the computational resources spent in executing it. Therefore the total cost of local computation could be resumed in the formula available in 2.16 consisted of adding the results of multiplying the delay of local task execution by the weight of time consumption, with the multiplication of the energetic cost of executing a task by the weight of those energetic costs. Both weights had to sum up to a value of one, and each could never be less than zero.

$$(TC)_n^{lc} = W_n^t * (T)_n^{lc} + W_n^e * (E)_n^{lc} \tag{2.16}$$

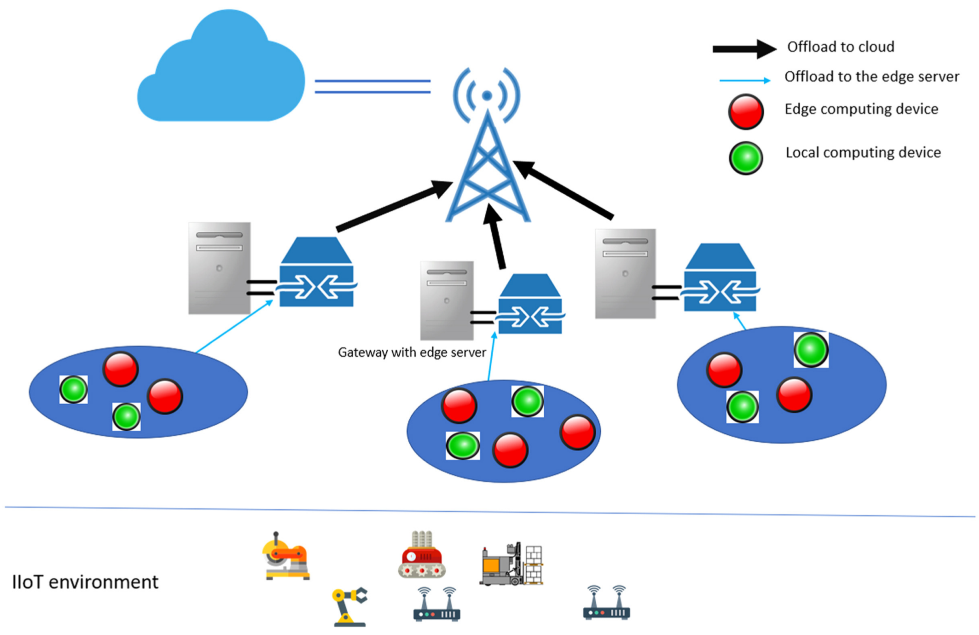


Figure 2.10: IIoT System Model for [20] (Taken from [20])

$$(T)_n^{ec} = (T)_n t^{ec} + (T)_n b^{ec} \tag{2.17}$$

$$(E)_n^{ec} = (E)_n t^{ec} + (E)_n b^{ec} \tag{2.18}$$

$$(TC)_n^{ec} = W_n^t * (T)_n^{ec} + W_n^e * (E)_n^{ec} \tag{2.19}$$

Therefore the sum cost of all devices in the edge offloading network could be derived as:

$$(SC)_{all} = \sum_{n=1}^{N} (1 - \alpha_n (TC)_n^{lc}) + \alpha_n (TC)_n^{ec}) \tag{2.20}$$

Where $\alpha_n$ represented the offloading decision for a device to whether execute a task locally, representing a value of $\alpha_n$ of zero, or whether to offload it to an Edge node for it to be executed there, representing a value of $\alpha_n$ of one.

To solve the problem at hand, it needed to be decomposed and solved to minimize the cost to achieve the optimal result, resulting in the following objective function and constraints:

$$
\begin{aligned}
min(\Gamma, f) \sum_{n=1}^{N} (1 - \alpha_n (TC)_n^{lc}) + \alpha_n (TC)_n^{ec}) \\
\text{subject to} \\
\text{C1:} \alpha_n \in (0,1), \forall n \in N \\
\text{C2:} (1-\alpha_n)(TC)_n^{lc} + \alpha_n (TC)_n^{ec} \leq \tau_n, \forall n \in N \\
\text{C3:} 0 \leq L_n \leq F, \forall n \in N
\end{aligned}
\tag{2.21}
$$

$\Gamma$ represented the vector that contained the decision whether to offload a task or not, and $f$ represented the vector that contained the allocation of computational resources. When it came to constraints in the problem definition, C1 defined that end devices could choose to either execute the tasks locally or offload them to be executed on an Edge node, as previously stated, C2 defined that task execution delay should be kept within a constant and tolerable value, independently of the type of node, local or Edge, where it was executed, and C3 defined that the available resources at the Edge server were always higher than the ones assigned for a number of Edge devices. This formulation required thinking of an adequate Reinforcement Learning strategy, concretely a Q-Learning strategy. With this strategy, a policy informed an agent what actions happened and the circumstances that made those actions occur by assigning rewards to the state transitions depending on how much they benefited the system, finding the optimal action-state selection policy. For this purpose, a matrix, named Q-Table, was used to store, for each activity at each state, the most extreme anticipated future rewards. For each column, an action was recorded in the Q-Table, while for each row, the states would be represented. The process was divided into iterations, where in each of the rewards associated with the actions leading to the states would be improved, which effectively meant that the more iterations that occurred, the better results the mechanism would obtain. Given that IoT devices select data for offloading based upon the state of the system, the energy level consumption of the system, and data transmission rate, the authors decided to monitor these factors for the IIoT system devices, and since system state was seen as a combination of two components, it was possible to derive a Q-Learning policy that would allow achieving the optimal solution. Concretely the two components which constituted the system's state, $t_c$ which was the total sum cost ($SC_{all}$) and $ac$, which was the result of subtracting all of the Edge servers' resources, with the sum of all allocated Edge servers' resources for the execution of each task. In terms of actions, they were stored so that all the information regarding offloading decisions and resource allocation for each task was easy to access and, therefore, easier to obtain the optimal policy.

$$
\begin{aligned}
state = (t_c, ac) \\
ac = F - \sum_{n=0}^{N} f_n
\end{aligned}
\tag{2.22}
$$

To run simulations, it was defined that an Edge server would be set at the center, the User

Equipment devices randomly distributed across the network at less than 200 meters of the central server node, while tasks had a size, in Kilobytes, between 300 and 500, and the assigned weights to time and energy consumption, were 0.5, for each. Edge servers' computation resources were equally distributed among the user devices to run and to understand the results better, the proposed mechanism was tested against a simpler one, where tasks were either fully offloaded or completely executed locally. As visible in Figure 2.11, the Q-Learning method displayed better sum cost, and in the initial stages where not many user devices were in the system, the full offloading curve was higher than the Q-Learning curve, since Edge nodes' could not offer computational resources for all devices that offload their tasks, while the local offloading sum cost curve increased immensely as the number of devices in the system increased. The figure also shows the sum cost for the Q-Learning, full offloading and local offloading, in relation to the capacity of the Edge servers, in GHz/s, and once again, the Q-Learning mechanism displayed the best result, while the local offloading curve never changed, given that executing tasks locally does not impact resources in Edge nodes. The other curves verified a decrease when the computational resources of Edge servers decreased, given a decrease in time of execution as more resources were allocated to user devices. As expected, the results showed that in terms of reward calculation, the Q-Learning performed greatly as the reward increased, when the number of time steps increased.

It was possible to conclude that the Q-Learning mechanism proposed greatly improved resource spending in task offloading when compared to simpler, more direct approaches. However, the network used for simulations was relatively simple, and understanding was still lacking, regarding how this approach would perform under a more complex network structure.

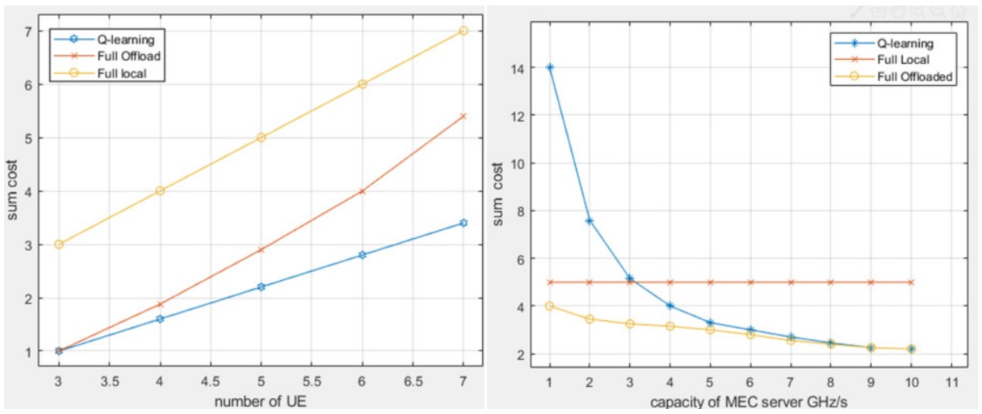


Figure 2.11: Results of simulations, showing relation between sum cost and number of user devices in the system and the relation between sum cost and Edge resource usage(Taken from [20])

## 2.5 Summary

This section presented the State of the Art of proposed IoT/IIoT task offloading mechanisms. Even if some papers available had to be filtered due to time constraints, paper quality, and primarily due to the infeasibility of having every single paper written presented, it was still possible to come up with a key set of findings. Though the papers presented differed in terms of techniques and concepts that motivate and justify the methodologies used, it was clear that all these papers resulted from the need to understand if one can improve resources needed to perform offloading tasks. In all the presented cases, this was confirmed as resources spent for energy, time, storage, availability were successfully diminished, in the process, while guaranteeing that the mechanism itself performed accordingly. It is important to mention that the difference in number between papers that proposed IoT oriented solutions and those that presented IIoT oriented solutions justified dividing the second chapter in the manner presented since it is possible to adapt or to even emulate many of the solutions found for the IoT and bring them into IIoT environments. Below a table is presented that summarizes some of the papers presented and the results obtained and categorizes them regarding types of technologies used, type of environment where it was tested, offloading criteria defined, and issues or future work.

Table 2.1: Literature Review Summary

| Reference | Area | Proposed System | Offloading Criteria | Research Area | Results | Testing Environment | Issues/Future Work |
|---|---|---|---|---|---|---|---|
| [14] | IoT | Multi-population cooperative elite algorithm, based on Genetic Algorithms. | Execution delay and energy consumption of applications. | Artificial Intelligence | Reasonable allocation of resources and overhead reduction. Superior to the tested Random, Greedy, and Standard Generic algorithms. | Laboratorial Experimentation | Take Cloud into consideration, consider an Edge-Cloud collaboration architecture. Research should also focus on user mobility and dynamic computation offloading. |
| [22] | IoT | Nature-inspired meta-heuristic schedulers (ACO and PSO) used to propose two different scheduling algorithms. | Communication cost and response time. | Artificial Intelligence | ACO and PSO task offloading algorithm improved response time significantly, though the ACO performs better and could balance node load more efficiently. | Laboratorial Experimentation | Include power consumption, communication cost, and computation cost in the offloading criteria of the mechanisms and account for collaboration between IoT sensors. Dynamic scenarios, where the produced data rate and sensor nodes change dynamically. |
| [16] | IoT | Distributed predictive offloading scheme for multi-tiered fog computing systems. | Time-average power consumptions, stability of each queue in the system. | Artificial Intelligence | Scheme achieves a tunable power-latency tradeoff, and could shorten latency even when future information is only mildly present, even when there exist prediction errors. | Laboratorial Experimentation (though there is a possibility of applying the scheme in real-world use cases). | Model can extend to accommodate general settings so that wireless channel states can be known instantaneously. |

Table 2.1: Literature Review Summary

| Reference | Area | Proposed System | Offloading Criteria | Research Area | Results | Testing Environment | Issues/Future Work |
|---|---|---|---|---|---|---|---|
| [25] | IoT | Two cooperative bargaining game algorithms - Tempered Aspirations Bargaining Solution (TABS) and Gupta-Livne Bargaining Solution (GLBS). | TABS is used for time-sensitive offloading services, GLBS used for computation-oriented offloading. | Artificial Intelligence | Algorithms could fairly distribute system resources through all devices while satisfying their fair-oriented axioms and maintained appropriate performance balance. Under dynamic network system it could provide flexibility, responsiveness to network system conditions and adaptability. | Laboratorial Experimentation (though the algorithms can be considered appropriate to work in the real world). | Work on possible privacy issues during the offloading process, understand if mobile devices can adapt to the dynamic network environments. Careful and additional investigation regarding information exchange and overhead in communications is needed. Extension of scenario from a singular cloudlet Fog node to multiple. Take into consideration interference management, control overhead, and load balancing. |
| [28] | IoT | Framework for optimal Task Offloading (system's IoT devices have Edge server and Cloud server coverage). | Transmission delay, energy consumption, and resource allocation. | Functional Programming given that the "optimal" offloading strategy defined was obtained by comparing completion times of tasks in IoT devices and the Edge server. | Number of completed tasks was improved, and there was a reduction of energy consumption observed with the proposed solution. | Laboratorial Experimentation. | Enable content caching in Edge nodes, mobility-aware allow for cooperative and mobility aware Edge node caching and address possible security and privacy concerns. |

Table 2.1: Literature Review Summary

| Reference | Area | Proposed System | Offloading Criteria | Research Area | Results | Testing Environment | Issues/Future Work |
|---|---|---|---|---|---|---|---|
| [15] | IoT | AutoScaler (functions as a front-end for the offloading architecture proposed, that besides the front-end, consists of a back-end and an offloading simulator capable of generating dynamic offloading workload of multiple devices). | Response time (offloading request), handling capacity regarding multiple offloading requests, and capacity to distribute said requests to different instance types. | Cloud computation (usage of Amazon EC2) and Functional Programming (components in Java language). | For metrics, the time to distribute requests between servers improved, even as load increased, though this phenomenon provoked a CPU resource usage, limiting the number of tasks that were possible to execute parallel to each other. For performance, AutoScaler introduced extra-time to the time necessary to respond to an offloading request, in about one-hundred and fifty milliseconds, though it is possible to improve on this value (processing tasks in Cloud/managing tradeoff between the price for utilization and computational resources of the server chosen for offloading). | Laboratorial Experimentation (though it is possible to extrapolate the solution to real-world use cases). | Study cost of building such a system in a real-world scenario. |
| [17] | IIoT | Two-stage heuristic mechanism. | Average and worst-case delay performance of the system (delay jitter). | Mathematical and Economics based solution | Proposed mechanism was successful in controlling the risk of intense delay jitter. | Laboratorial Experimentation (more mathematical resolution than programming one). | - |
| [11] | IIoT | Energy-efficient, gradient algorithm-based, computation offloading scheme. | Energy consumption(tasks had to be executed within a desired delay and energy overhead). | Artificial Intelligence. | Energetic consumption was improved on local computing nodes, and overall, the completion time was improved, and delay constraint was guaranteed. Nonetheless, the energetic cost of task transmission to the Cloud increased. The more iterations of the algorithm executed, the more likely it was to find the optimal value to achieve minimum energy consumption. | Laboratorial Experimentation. | Integrate Deep-Learning and green framework concepts to make decisions "greener" and smarter. |

Table 2.1: Literature Review Summary

| Reference | Area | Proposed System | Offloading Criteria | Research Area | Results | Testing Environment | Issues/Future Work |
|---|---|---|---|---|---|---|---|
| [12] | IIoT | Pattern-identified online task scheduling mechanism for the networking infrastructure in a multi-layered Edge system. | Computation performance and energy consumption. | Artificial Intelligence/Optimization. | When compared to a greedy algorithm and an offline Hungarian method algorithm, the proposed mechanism reduced latency in the processing of tasks and performed better in terms of time consumption and average decision-making time while reducing the number of execution errors present in shorter deadlines tasks. | Laboratorial Experimentation. | Consider individual requirements of IIoT applications, and verify them through the usage of datasets to achieve optimal performance for task handling. A new approach, based on game-theory should be studied to develop a distributed computational framework for Edge computing. |
| [19] | IIoT | Distributed game-theoretic Quality of Service-aware computation offloading mechanism. | Computation time and energy consumption. | Artificial Intelligence. | Mechanism proposed was more efficient than the other tested algorithms (greedy and simulated annealing) while being stable when dealing with tasks that were intensive computationally. The more IIoT devices present in the process, the more effective the proposed multi-hop algorithm was. | Laboratorial Experimentation. | Consider individual requirements of IIoT applications, and verify them through the usage of datasets to achieve optimal performance for task handling. Perform studies on a nonelastic Cloud environment and adopt a bandwidth-sharing model to enable throughput allocation according to data size or urgency of tasks. |
| [20] | IIoT | Edge computation, reinforcement learning-based, task offloading mechanism. | Latency and power consumption. | Artificial Intelligence. | Mechanism displayed more efficient resource consumption for user devices and Edge servers when compared to just executing tasks locally or just offloading them to the Edge nodes to be executed there. | Laboratorial Experimentation. | Attend to more complex systems and offloading criteria. Utilize deep Q-Learning, to improve resource usage. |

In the next chapter, the problem formulation will be extensively detailed.

# Chapter 3

# Problem Formulation

In the current chapter, the formulation of the problem at hand is described to understand the mathematical logic behind the development of the algorithms.

## 3.1 Problem Formulation

This section will explain the mathematical theory behind the algorithms developed according to the parameters and metrics on which the task offloading mechanism was based, such as CPU resources, memory resources, and associated cost. The section itself will be divided into three subsections, one for each algorithm created.

### 3.1.1 Greedy Algorithm

The Greedy algorithm, which focused solely on a device's available resources, made use of a file in which information regarding a device's measured parameters was stored. Namely, its CPU's frequency, CPU usage, memory available for usage, percentage of memory usage, and associated cost of offloading a task to said device were stored. This file was generated by connecting the task offloading mechanism to a distributed platform, named DINASORE (Dynamic INtelligent Architecture for Software and MOdular REconfiguration) and performing a single measure of these metrics before any Function Block (FB) was associated to a device. This decision was made since the optimizer would use the data to perform calculations under the assumption that devices were running at optimal conditions, when executing all algorithms. This assumption held for all the other algorithms since they utilized the same devices' input files for calculations. Mathematically speaking, the Greedy algorithm utilized a sum obtained as presented in equations 3.1 and 3.2. In the first of both equations presented, the device's CPU frequency value was added to multiplying 1000 to the device's available CPU percentage. In the second, the operations are similar, but the parameters were different. Namely, they reflected a device's memory parameters, such as its available memory and its available memory percentage. Both percentage values were multiplied by 1000, so that they were "converted" to an order of magnitude adequate to the device's CPU frequency and available memory, respectively. It should be noted that both the Greedy and Improved Greedy algorithms prioritized only one metric at a time, which could chosen by the user, either the CPU parameters, or the memory parameters, but never both simultaneously.

$$tdCPUCap = dCPUFreq + ((100 - dCPUPerc) * 1000) \tag{3.1}$$

| Equation 3.1 Abbreviations | |
|---|---|
| **Abbreviation** | **Meaning** |
| tdCPUCap | Total Device CPU Capacity |
| dCPUFreq | Device CPU Frequency |
| dCPUPerc | Device CPU Percentage |

$$tdMEMCap = daMEM + ((100 - dMEMPerc) * 1000) \tag{3.2}$$

| Equation 3.2 Abbreviations | |
|---|---|
| **Abbreviation** | **Meaning** |
| tdMEMCap | Total Device Memory Capacity |
| daMEM | Device Available Memory |
| dMEMPerc | Device Memory Percentage |

### 3.1.2 Improved Greedy Algorithm

The Improved Greedy algorithm is, as the name suggests, an improvement to its Greedy counterpart, as it takes into consideration not only a device's available resources but also the toll a Function Block's execution takes on its attributed device. Therefore, in Mathematical notation, the Improved Greedy algorithm utilized a subtraction of two different sums. As specified in equation 3.3, the total device capacity for the CPU parameter was subtracted to the measured CPU percentage consumption value, verified when a device executed the Function Block. In contrast, for the memory parameters, the device's total memory capacity was subtracted to the sum of the available memory after the Function Block had been executed, with the measured memory percentage consumption value, as seen in 3.4. Both of the percentages were multiplied by 1000 to make the order of magnitude adequate for the whole equation.

$$fbCPUNe = tdCPUCap - ((aFbCPUPerc) * 1000)) \tag{3.3}$$

| Equation 3.3 Abbreviations | |
|---|---|
| **Abbreviation** | **Meaning** |
| fbCPUNe | Function Block CPU Necessities |
| tdCPUCap | Total Device CPU Capacity |
| aFbCPUPerc | Average Function Block CPU Percentage |

$$fbMEMNe = tdMEMCap - (aFbAvaMEM + (aFbMEMPerc * 1000)) \tag{3.4}$$

Equation 3.4 Abbreviations

| **Abbreviation** | **Meaning** |
|---|---|
| fbMEMNe | Function Block Memory Necessities |
| tdMEMCap | Total Device Memory Capacity |
| aFbAvaMEM | Average Function Block Available Memory |
| aFbMEMPerc | Average Function Block Memory Percentage |

### 3.1.3 Q-Learning Algorithm

In mathematical terms, the Q-Learning algorithm followed the equation presented in 2.9. For the experiment, the adopted values for the discount and learning rates were 0.99 and 0.1, respectively. The Q-table corresponds to a matrix where the number of rows represented the number of devices present in the system. The number of columns represented the Function Blocks executed in the system. At the end of the algorithm's execution, a display would allow the user to understand the level of adequacy of running a specific Function Block for each device. The reward obtained at each step was considered a sum of the evaluation of different metrics, such as CPU frequency, CPU and memory percentage, available memory, and offloading cost. As it is possible to observe from equations 3.5 to 3.9 the process of obtaining a reward value at each step is very complex and involved an individual evaluation of those different metrics, namely for the CPU frequency, its value was obtained by attributing a higher reward to devices that displayed higher frequency, for the CPU and memory percentages the evaluation was done by subtracting the device's available metric percentage to the monitored metric percentage the Function Block in average needed to perform correctly. The result of this subtraction could either be a negative number, indicating that the device would not be an ideal place to execute that Function Block. In contrast, a positive number would show precisely the opposite. The result itself was evaluated by giving a bigger reward the more significant the positive difference obtained. For example, if the result of the subtraction was 50, the reward was 50 as well, while if the result was -50, the reward would be -25. The maximum difference allowed was 100, both for a negative and positive result. Similarly, the available memory parameter evaluation tested if the device had enough available memory to make up for the necessary Function Block's memory requirements. This was achieved by dividing the device's available memory with the Function Block's execution's average memory usage. If the value was less than 1.00, there was no memory available to execute the Function Block. Therefore the reward given in that scenario was -25, while if the division's result was precisely one, the reward was 0. For results between 1.00 and 2.00, the reward was increased by ten as the result increased by .10, for example, for a 1.10 result, the reward was 10, for 1.20, the reward was 20, for 1.30, the reward would be 30, and so forth until the maximum reward of 100 for a result of 2.00. For the final step, the reward value obtained in the previous calculations was subtracted, the same reward value times the device's offloading cost. This acted as a sort of "tax" for having to offload the task to the device itself and would only reflect alterations on the reward value if the offloading cost was greater than zero, for example, in the case of a Cloud device.

$$Reward = CPUfreqeval + MEMavaeval + percEval - cOffEval \tag{3.5}$$

$$CPUfreqeval = eval(CPUfreq)$$
$$where$$
$$eval(x) = \begin{cases} 0, & \text{if } x <= 0 \\ 10, & \text{if } x <= 5000 \\ 20, & \text{if } x <= 10000 \\ 30, & \text{if } x <= 15000 \\ 40, & \text{if } x <= 20000 \\ 50, & \text{if } x <= 25000 \\ 60, & \text{if } x <= 30000 \\ 70, & \text{otherwise} \end{cases} \tag{3.6}$$

$$MEMavaEval = eval$$
$$(avaDevMEM, fbAvaMEMres, desFact),$$
$$where$$
$$desFact > 1.00$$
$$and$$
$$desFact < 2.00$$
$$eval(x,y,z) = \begin{cases} 0, & \text{if } x/y <= 1.00 \\ 1, & \text{if } x/y >= z \\ 2, & \text{otherwise} \end{cases} \tag{3.7}$$

$$PercentageEvalution = eval$$
$$(devCPUPerc, fbCPUPerc, targValue)$$
$$or$$
$$eval(devMEMPerc, fbMEMPerc, targValue)$$
$$where$$
$$targValue > -100$$
$$and$$
$$targValue < 100$$
$$eval(x,y,z) = \begin{cases} 0, & \text{if } x-y <= -100 \\ 1, & \text{if } x-y <= z \\ 2, & \text{otherwise} \end{cases} \tag{3.8}$$

$$cOffEval = reward - (reward * devCOff) \tag{3.9}$$

| Equations 3.5, 3.6, 3.7, 3.8, 3.9 Abbreviations | |
|---|---|
| **Abbreviation** | **Meaning** |
| CPUfreqEval | CPU Frequency Evaluation |
| MEMavaEval | Memory Available Evaluation |
| percEval | Percentage Evaluation |
| cOffEval | Cost of Offloading Evaluation |
| CPUfreq | CPU Frequency |
| avaDevMEM | Average Device Memory |
| fbAvaMEMres | Function Block Average Memory Resources |
| desFact | Desired Factor |
| devCPUPerc | Device CPU Percentage |
| fbCPUPerc | Function Block CPU Percentage |
| targValue | Target Value |
| devMEMPerc | Device Memory Percentage |
| fbMEMPerc | Device Memory Percentage |
| devCOff | Device Cost of Offloading |

In this chapter the problem formulation was explained, and in the following chapter the process of implementation will be detailed.

# Chapter 4

# Implementation

In this chapter, the task offloading mechanism implementation is further detailed. The chapter itself will be divided into different sections. The first section explains the mechanism architecture, the second section explains file operation-related code, and the third section presents the algorithms in detail.

## 4.1 Mechanism Architecture

This section describes how the task offloading mechanism was designed, and the system that was thought out for the experiment is explained. The algorithms' initial conception will be presented. There will be a brief discussion about the technology used.

### 4.1.1 Mechanism Design

To understand how the mechanism was designed, it is necessary to add more details behind DINASORE's functioning. DINASORE allows for code execution within the CPSs through Function Blocks that abstract software and hardware modules suitable for manufacturing requirements. [31] To configure and deploy these Function Blocks, the 4DIAC-IDE should be utilized since it proves capable of communicating with OPC-UA (Open Platform Communications-Unified Architecture) servers, which is the main gateway to devices running DINASORE. Every instance of DINASORE has exactly one instance of an OPC-UA server running associated with it. 4DIAC-IDE is an integrated development environment designed to allow for the creation and deployment of Function Blocks, and compliant with the ldquoFramework developed for distributed industrial automation and control (4DIAC), and that was based on the IEC(International Electrotechnical Commission)-61499 standard, which was developed for distributed, modular, and flexible control systems. [35]

OPC-UA is also a standard for industrial automation that allows for data exchange and interoperability from lower levels, such as sensors and actuators, to higher levels, such as control and communication systems, for example, central servers and Clouds. [13]

In DINASORE's specific case, Function Blocks are programmed in the Python language and XML (Extensible Markup Language) since XML tags encode the metadata related to the Function Block itself. This fact justified the choice of the usage of the Python language in the development of the task offloading mechanism, alongside other factors, such as the language's popularity and how flexible it is in developing Artificial Intelligence solutions.

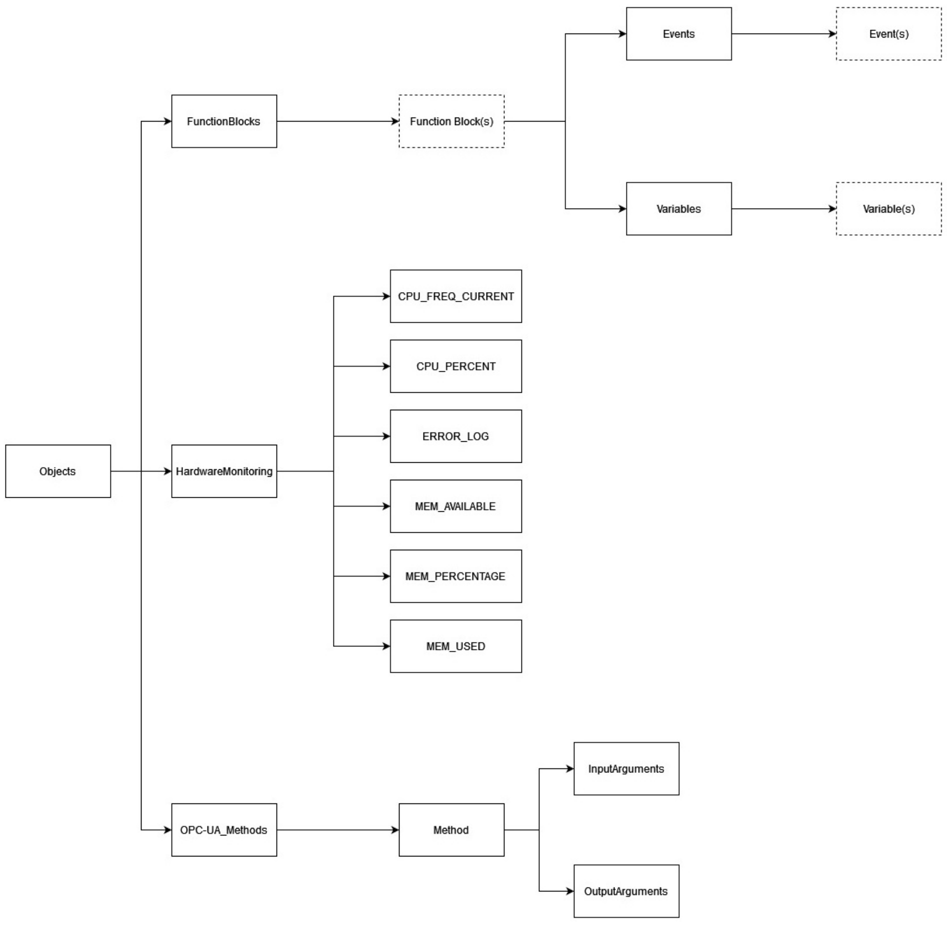
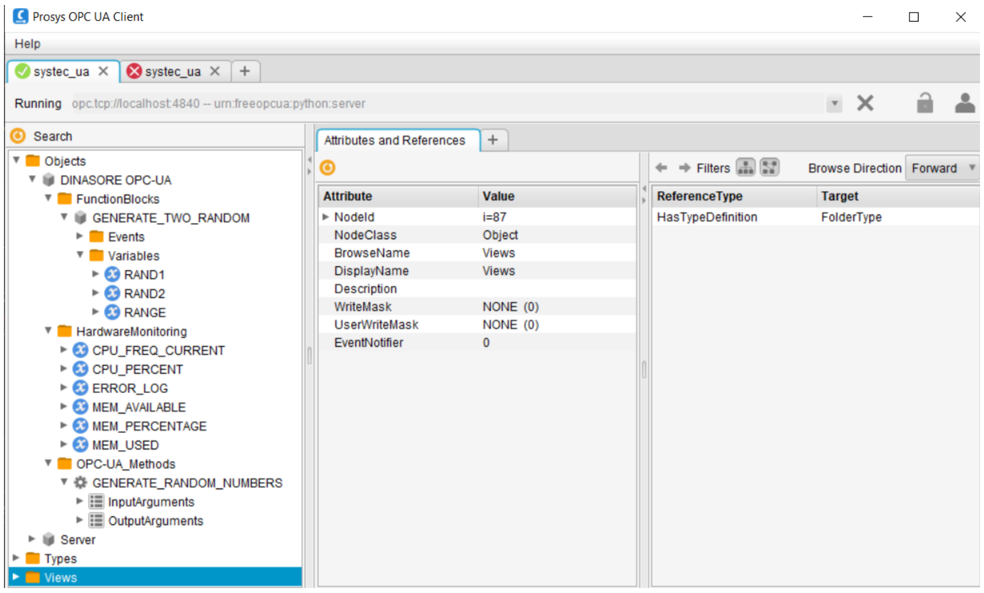


Figure 4.1: OPC-UA Model

But if in the scope of the research, the implementations of the Function Blocks were unimportant, when compared to the task they were performing, it is more important to describe the OPC-UA model since it would be required to understand the levels of resource consumption during the execution of Function Blocks, and the OPC-UA server associated with the DINASORE instance allows for this information to be easily obtainable. The OPC-UA model is presented in figure 4.1 and is structured in three main folders, the Function Blocks folder, where information regarding the Function Blocks and their execution parameters are stored, the Hardware monitoring folder, which holds a set of variables that are used to monitor specific device resources while running the respective DINASORE instance and the OPC-UA method folder, that is only populated if the user wrapped a portion or the whole pipeline within a method wrapper. This information can be seen by using an OPC-UA client such as Prosys, which is capable of reproducing it as presented in figure 4.2.



| Attribute | Value |
|---|---|
| NodeId | i=87 |
| NodeClass | Object |
| BrowseName | Views |
| DisplayName | Views |
| Description | |
| WriteMask | NONE (0) |
| UserWriteMask | NONE (0) |
| EventNotifier | 0 |

| ReferenceType | Target |
|---|---|
| HasTypeDefinition | FolderType |

Figure 4.2: Prosys OPC-UA Data Display

Considering these points, it was clear that for the mechanism to perform adequately, it was necessary to decide how it would communicate with devices running DINASORE instances. Namely, two options were available, either the mechanism would perform its tasks by monitoring the execution of the Function Blocks and by monitoring the different devices' resources and use that data to provide a possible optimization solution to the user as a means of suggestion, given that it wouldn't be possible to dynamically reconfigure the DINASORE configurations, instead leaving that option to the user. The other option would be to allow for dynamic reconfiguration of the DINASORE instances by using the mechanism as an interface between the 4DIAC-IDE and the different DINASORE instances. But even if this option would be better from the users' standpoint, to make that approach work, it would be necessary to alter DINASORE's code. Therefore, due to time constraints and because changing DINASORE's source code was outside of the scope of the research, the mechanism was designed as presented in figure 4.3. It comprised of three major parts, the 4DIAC-IDE, the DINASORE instance, and the optimizer, which by itself needed to have interfaces implemented, capable of communicating with both the 4DIAC-IDE and the DINASORE instances to have full access to all the data being monitored along with the execution of a pipeline and to present the user the proposed optimization solution. Specifically, both the DINASORE instance and the mechanism measured the different parameters and metrics chosen to influence the mechanism and stored the readings in text files, which would function as inputs to the mechanism.

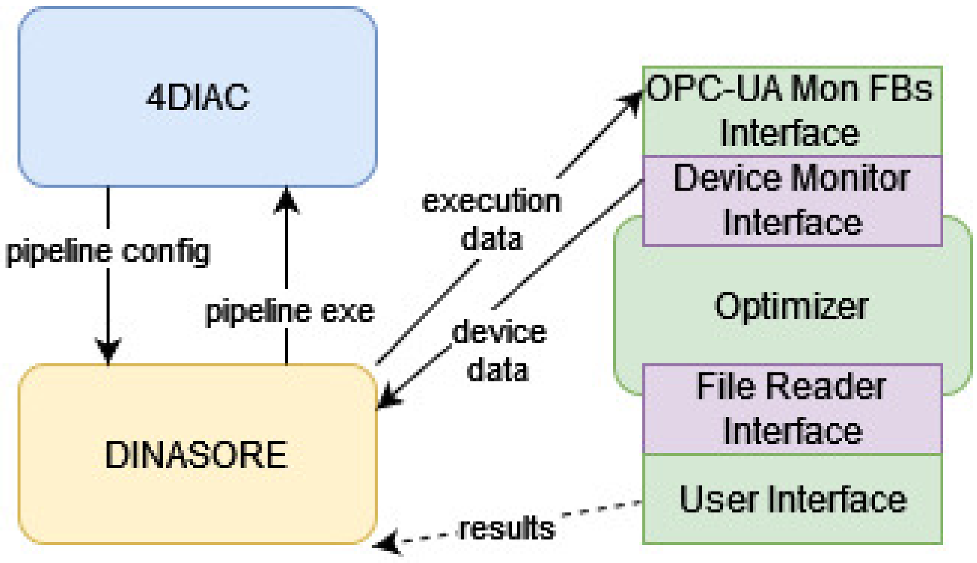


Figure 4.3: Task Offloading Mechanism Design

#### 4.1.1.1 Class Diagram

Before explaining the code and algorithms, it is necessary to present the class diagram, which describes the whole mechanism's two main entities. As seen in figure 4.4, those two entities are devices and Function Blocks. The devices were characterized by having a name, a CPU frequency, a CPU usage percentage, available memory, memory usage percentage, and associated cost. In contrast, Function Blocks had as parameters a name, the calculated average CPU usage percentage, the calculated average memory usage percentage, and the calculated average memory resources spent. Other optional parameters included the event and argument list. Although they weren't utilized in the scope of the experiment, their presence allowed for some code future-proofing. The code developed for the models can be seen in figures 8.1 and 8.2.

#### 4.1.1.2 Algorithm Conception

Though they will be further expanded in the next chapter, the thought process that originated the algorithms needs to be referred to. In the conception of the algorithms, it was necessary to define the system metrics in which the offloading mechanism had to focus the most. The metrics selected

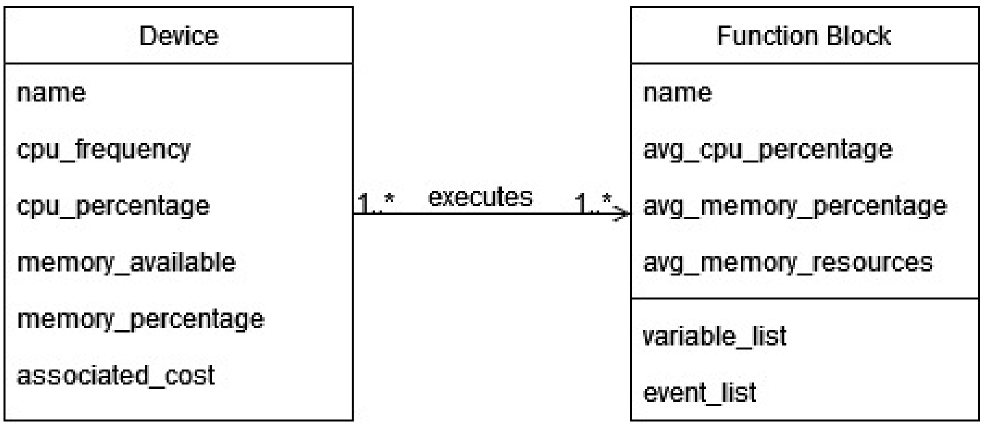


Figure 4.4: Task Offloading Class Diagram

were CPU resources, memory resources, and offloading cost. Time metrics weren't reflected in the offloading parameters, since the pipelines executed on a loop until the user killed the processes associated with these executions, manually.

The first algorithm, named the Greedy algorithm, performed calculations based exclusively on a device's available resources without considering Function Block consumption or toll on a given device to present to the user a possible optimization solution. The second algorithm, named Improved Greedy algorithm, was, as the name suggests, similar to the Greedy algorithm. Still, in this case, it did consider Function Block needs for devices, though in a simplistic manner. The final algorithm, a Q-Learning algorithm, performed calculations by taking the three metrics into consideration. It would better understand Function Block consumption or toll on a given device but also understand, for example, if tasks should be performed on a local or Edge level or to be executed at the Cloud level, guaranteeing that a Cloud execution had an associated cost which could play an important part in the selection of devices on which tasks were to be executed.

One important note to refer to is that Cloud solutions, considered to be "professional" such as Amazon AWS, Digital Ocean, or Google Cloud, offer very expensive solutions for systems with very high CPU, RAM, and storage specifications. Since a device with some capacity was required to come to a conclusion about the importance, or lack thereof, of including a Cloud level in a task offloading mechanism, and given the costs of acquiring a solution like the ones aforementioned are prohibitive, a decision was made in conjunction with the DIGI2 lab, that one physical device would be utilized as a representative of the Cloud level device needed for the research.

#### 4.1.1.3 Technologies

To develop the mechanism, the Python language was chosen. It is a high-level and general-purpose language, and it is one of the most popular and utilized languages by developers. Its advantages include improved code readability, support for multiple programming paradigms, an open-source community that provides many useful libraries, and beginner and user-friendliness. [34]

Specifically, the version that was chosen to develop the solution was version 3.7.9, as it was necessary to utilize a version compatible with the OpenAI-Gym library. Other libraries, including NumPy and opcua, were used. NumPy is the main array programming library for the Python language and is utilized to advance research in fields such as physics, chemistry, engineering, finance, and economics [18]. The opcua library is the Python implementation of the OPC-UA stack and its tools, while the OpenAI-Gym library is a toolkit for Reinforcement Learning research [10].

An MQTT (Message Queuing Telemetry Transport) broker was used, named Mosquitto, and the paho-mqtt library was used to establish communication between MQTT servers and clients. MQTT will be further detailed in the following chapter.

## 4.2 Algorithms and File Operations

This section will present the implementation details of both the algorithms developed and the classes that were developed for file manipulating reasons.

### 4.2.1 File Operations

The classes developed for file operations were coded to centralize the process of reading the data provided by DINASORE's monitoring module and storing it for algorithmic manipulation. Namely, six classes were defined. The first, FileReader, functioned as an auxiliary class, responsible for simply reading the lines of a specified file and storing them in a list. The second class, FunctionBlockReader, read the contents of the Function Block history file generated by DINASORE for each machine it executed on, which contained the name of the Function Blocks that were executed and initial and ending timestamps in which that Function Block executed. In this class, two dictionaries, the Python version of key and value collections, were instantiated. To group the information regarding all of a Function Block's initial and final timestamp in the FB history dictionary and start storing information regarding Function Blocks that executed instantaneously, in the monitor info dictionary. In the cases where the initial timestamp was equal to the final timestamp, consumption was considered 0, as no resources were assumed to be spent for executing that Function Block at that particular time. The monitor info dictionary was also used in another class, MonitorReader, where the file that stored the result of the monitored resources was read. Its contents were utilized to establish a match between the consumption verified in a Function Block execution, and the monitoring's measured timestamp. This was done by verifying if the monitored timestamp was between the interval in which a certain Function Block executed. If this assumption held, then the information regarding resource consumption for that Function Block was stored in the monitor info dictionary. The DeviceAvailabilityReader would read the file generated by the task offloading mechanism after connecting to the correct DINASORE OPC-UA server instance and performing a single reading of the resources available. The information was maintained in a list so that it was easier to separate between the different devices running DINASORE. Finally, the DataConnector class was the class that centralized all the information loaded into the monitor info and FB history dictionaries and took the data available in the monitor dictionary to calculate the average resources spent in each Function Blocks' execution. To achieve this, the measured values list associated with each Function Block was iterated, and a sum of the obtained CPU usage percentage, memory usage percentage, and memory consumed was calculated and divided by the total amount of readings verified and traced back to the Function Block itself. Figures 8.3 to 8.7 present the code of the classes, by the same order in which they were mentioned.

### 4.2.2 Algorithms

The algorithms will be explained in the next sub-subsections. The first algorithm discussed will be the Greedy, followed by the Improved Greedy and the final algorithm will be the Q-Learning algorithm.

#### 4.2.2.1 Greedy Algorithm

For executing the Greedy algorithm, a dictionary was instantiated. This dictionary would store the points attributed to each device to understand how they stood regarding the chosen parameter to be prioritized when compared with each other. Depending on the parameter chosen to be prioritized, either CPU or memory resources, the mechanism would compare the device parameters by iterating through the list of all devices connected to the DINASORE and, consequently performing this comparison between all of the devices present. Considering a scenery where a device X was compared to a device Y, if device X scored better when evaluated accordingly to either equation 3.1 or 3.2, for the metrics each equation represents, then device X would be attributed 1 point. In contrast, device Y would have a point subtracted. If the opposite occurred, device Y would score a point, while device X would lose it. In case of a tie, both devices would be awarded a point. After all the comparisons were made, a list was displayed to inform how each device fared for the metric chosen by the user to be optimized. For that, an auxiliary class named ShowResult was utilized, allowing for reuse with the Improved Greedy algorithm.

**Algorithm 1** Greedy Algorithm

```
procedure RUNGREEDYALGORITHM(deviceList, optionToPrioritize)
    if $optionToPrioritize = 1$ then
        prioritizeResources(deviceList, "cpuFrequency", "cpuPercentage")
    else if $optionToPrioritize = 2$ then
        prioritizeResources(deviceList, "memoryAvailable", "memoryPercentage")
    end if
```

**Algorithm 2** Prioritize Resources Method

```
procedure PRIORITIZERESOURCES($deviceList$, $parameterToOptimize1$, $parameterToOptimize2$)
    $scoreMap \leftarrow newMap < deviceName, deviceScore >$
    for <X, Y in combinations(deviceList, 2)> do
        if $getParametersSum(X, parameterToOptimize1, parameterToOptimize2) > getParametersSum(Y, parameterToOptimize1, parameterToOptimize2)$ then
            scoreMap(X, deviceScore++)
            scoreMap(Y, deviceScore−−)
        else if $getParametersSum(X, parameterToOptimize1, parameterToOptimize2) < getParametersSum(Y, parameterToOptimize1, parameterToOptimize2)$ then
            scoreMap(Y, deviceScore++)
            scoreMap(X, deviceScore−−)
        else
            scoreMap(X, deviceScore++)
            scoreMap(Y, deviceScore++)
        end if
    end for
```

#### 4.2.2.2 Improved Greedy Algorithm

Coded similarly to its Greedy counterpart, the main difference of the Improved Greedy version is that when it came to the comparison of all the devices connected to DINASORE, related to the desired metric to be optimized, is that the parameters were evaluated accordingly to equations 3.3 or 3.4 respectively. The point attribution worked the same in both Greedy algorithms. To achieve this difference in calculations between the two algorithms, it was necessary to utilize different auxiliary classes. Namely, in the Utils class, which will be expanded upon in another section, two functions were utilized for each algorithm as a sort of central processing. As seen in Figures 8.12, 8.13 and 8.14, the methods compare_device_parameters and the improved_compare_device_parameters variant for the Improved Greedy algorithm are similar in the sense that they utilized almost all the same functions, namely some similarities include calling methods such as get_attribute_from_object, that returned the value of an attribute from an object, passed as an argument, for example, it could be utilized to get the value of the device's CPU frequency. Other functions were utilized, such as, add_point_to_device, and remove_point_from_device, which are self-explanatory. The difference between the comparison methods existed in the functions that were called to perform the evaluation of the metrics, namely, the get_parameters_sum, that represented the code version of equations 3.1 and 3.2. In contrast, get_improved_parameters_sum_memory method, represented the equation 3.4, and the get_improved_parameters_sum_cpu methods, that represented the equation 3.3.

**Algorithm 3** Improved Greedy Algorithm

```
procedure RUNIMPROVEDGREEDYALGORITHM($deviceList$, $optionToPrioritize$)
    if $optionToPrioritize = 1$ then
        prioritizeResourcesImproved(deviceList, "cpuFrequency", "cpuPercentage")
    else if $optionToPrioritize = 2$ then
        prioritizeResourcesImproved(deviceList, "memoryAvailable", "memoryPercentage", "avgMemoryResources", "avgMemoryPercentage")
    end if
```

**Algorithm 4** Prioritize Resources Improved Method

```
procedure PRIORITIZERESOURCESIMPROVED($deviceList$, $parameterToOptimize1$, $parameterToOptimize2$, $parameterToOptimize3$, $parameterToOptimize4$)
    $scoreMap \leftarrow newMap < deviceName, deviceScore >$
    for <X, Y in combinations(deviceList, 2)> do
        if $getImprovedParametersSum(X, parameterToOptimize1, parameterToOptimize2, parameterToOptimize3, parameterToOptimize4) > getImprovedParametersSum(Y, parameterToOptimize1, parameterToOptimize2, parameterToOptimize3, parameterToOptimize4)$ then
            scoreMap(X, deviceScore++)
            scoreMap(Y, deviceScore−−)
        else if $getImprovedParametersSum(X, parameterToOptimize1, parameterToOptimize2, parameterToOptimize3, parameterToOptimize4) < getImprovedParametersSum(Y, parameterToOptimize1, parameterToOptimize2, parameterToOptimize3, parameterToOptimize4)$ then
            scoreMap(Y, deviceScore++)
            scoreMap(X, deviceScore−−)
        else
            scoreMap(X, deviceScore++)
            scoreMap(Y, deviceScore++)
        end if
    end for
```

#### 4.2.2.3 Q-Learning Algorithm

The Q-Learning algorithm's code was divided into the QLearningAlgorithm class and the Environment class. The first class was where the algorithm itself was coded, while the second class represented the environment in which the algorithm was embedded. The Environment class was where the action space and the observation space were defined, namely the action space corresponded to a discrete space with the same length as the list of Function Blocks present system, while the observation space corresponded to a discrete space with the same length as the list of devices present in the system. It is in this class that the step function was coded, and it consisted of obtaining the current state of the Q-Learning algorithm, or the index of the device to be subject to evaluation, and passing it to the reward function alongside the action to be evaluated, corresponding to the Function Block to be analyzed. The reward function was composed of many other functions, that were the code reflection of what was explained from equations 3.5 to 3.9, and as seen in figure 8.17, it called the functions evaluate_percentage, to evaluate CPU and memory percentage usage, evaluate_available_memory, to evaluate available device memory with necessary memory to execute a Function Block, evaluate_cpu_frequency and measure_cost_of_offloading which are self-explanatory. The functions compare_resource_parameters and compare_memory_resources were utilized in the evaluate_percentage and evaluate_available_memory functions respectively and were the exact code representation of equations 3.8 and 3.7. The evaluate_percentage and evaluate_available_memory methods utilized them in the attribution of the value of the reward obtained, as exemplified in figures 8.19 and 8.20. The last methods written in the class included the render and the reset function, but if the first was a simple command-line print, given there was no complex GUI involved in presenting the results to the users, the second could be considered more relevant. The reset method was responsible for updating the state of the Q-Learning algorithm, which in specific terms meant that the function would increment the state value to assess other devices in the device list or reset the process by pointing again to the first device in the device list, as seen in figure 8.17.

The Q-Learning class, was responsible for executing the algorithm and initializing the Environment class since it needed to be aware of the environment to function correctly. The train model function would be where most of the code was developed. In it, the Q-Table was initialized alongside the learning rate, discount rate, exploration rate, and exploration rate decay parameters. One important thing to refer to is that the chosen number for episodes, or iterations, was 1000. The initial part of the method included obtaining the action to perform and passing it to the step function available to the environment to obtain the reward for it, while the following steps included the update of the Q-Table with the calculated value according to equation 2.9. After the calculation of the new exploration rate as displayed in figure 8.22, a new iteration would start. After all iterations were complete, a message with the results was shown to the user. The train method was called within another function, called exec, which was the one that would be called for executing the Q-Learning algorithm.

**Algorithm 5** Q-Learning Algorithm

```
procedure TRAIN($deviceList$, $functionBlockList$, $environment$)
    $stateSpaceSize := environment.getObservationSpace()$
    $actionSpaceSize := environment.getActionSpace()$
    $qTable := newMatrix[stateSpaceSize, actionSpaceSize]$
    $numEpisodes := 1000$
    $maxStepsPerEpisode := 10$
    $learningRate := 0.1$
    $discountRate := 0.99$
    $explorationRate := 1$
    $maxExplorationRate := 1$
    $minExplorationRate := 0.01$
    $explorationDecayRate := 0.01$
    for do$episodeinrange(numEpisodes)$
        $state := environment.getState()$
        $done := False$
        $rewardsCurrentEpisode := 0$
        for do$stepinrange(maxStepsPerEpisode)$
            $explorationRateThreshold = random.uniform(0, 1)$
            if then$explorationRateThreshold > explorationRate$
                $action; = maxIndexValue(qTable[state][])$
            else
                $action := environment.getActionSpaceSample()$
            end if
            done = $environment.step(action)$
            $qTable[state, action] = (1 - learningRate) * qTable[state, action] + learningRate * (reward + discountRate * np.max(qTable[newState, :]))$
            $state = new_state$
            $rewardsCurrentEpisode += reward$
            if then$done == True :$
                $break$
            end if
            $explorationRate = minExplorationRate + (maxExplorationRate - minExplorationRate) * exp(-explorationDecayRate * episode)$
        end for
    end for
end procedure=0
```

**Algorithm 6** Reward Function

```
procedure GETREWARD($deviceList$, $functionBlockList$, $state$, $action$)
    $reward := 0$
    $reward := evaluatePercentage(deviceList[state], functionBlockList[action], "cpuPercentage", "avgCpuPercentage") + reward$
    $reward := evaluatePercentage(deviceList[state], functionBlockList[action], "memoryPercentage", ""avgMemoryPercentage) + reward$
    $reward := evaluateAvailableMemory(deviceList[state], functionBlockList[action], "memoryAvailable", ""avgMemoryResources) + reward$
    $reward := evaluateCpuFrequency(state) + reward$
    $reward := measureCostOfOffloading(state, deviceList, reward) + reward$
```

**Algorithm 7** Cost of Offloading Method

```
procedure MEASURECOSTOFOFFLOADING($state$, $deviceList$, $reward$)
    $deviceAssociatedCost := getDeviceAssociatedCost(deviceList[state])$
    if $deviceAssociatedCost <= 0$ then
        return reward
    end if
    return (reward - (reward * deviceAssociatedCost))
```

### 4.2.3 Utility

The classes that formed part of the utility category include the Main class, the ClientConnector class, the ShowResult class, and the Utils class. The ShowResult class was utilized exclusively for displaying the results of the Greedy and Improved Greedy, and it consisted of simple prints to the console, as pictured in figure 8.9. While many methods of the Utils class have been presented, there are functions yet to be explained. While mostly related to file auxiliary methods, there was a method named reverse_percentage, which obtained the device's available CPU and memory percentages. This method was coded since what DINASORE measured. was the usage level of those respective parameters in the device. The methods that served as aides of file operations were, get_project_root, append_line_to_file, clear_file_contents and extract_file_info. While the first three are self-explanatory, the last one separated the values in the file from its parameter, which helped store data. The only method left to explain is initialize_dictionary_of_points, which was utilized for initializing the points dictionary for the Greedy and Improved Greedy algorithm. The other relevant class in this category is the ClientConnector, which, utilizing the python-opcua library, would be responsible for connecting to a DINASORE's OPC-UA server instance to perform a single read of a device's parameters to be stored in memory, that would serve as input for the mechanism. In this case, after the reading was executed, the data would be stored in a file to persist. This made it easier to draw some conclusion about the efficacy and accuracy of the mechanism. Finally, the Main served only to begin the whole process of data collection and algorithm execution as seen in figure 8.24.

In the next chapter the results will be presented and discussed.

**Algorithm 8** Evaluate Percentage Method

```
procedure EVALUATEPERCENTAGE($deviceInformation$, $functionBlockInformation$, $parameter1$, $parameter2$)
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 0) = 0$ then
        return -50
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -90.0) = 0$ then
        return -45
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -80.0) = 0$ then
        return -40
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -70.0) = 0$ then
        return -35
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -60.0) = 0$ then
        return -30
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -50.0) = 0$ then
        return -25
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -40.0) = 0$ then
        return -20
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -30.0) = 0$ then
        return -15
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -20.0) = 0$ then
        return -10
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, -10.0) = 0$ then
        return -5
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 0.00) = 0$ then
        return 0
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 10.0) = 1$ then
        return 10
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 20.0) = 1$ then
        return 20
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 30.0) = 1$ then
        return 30
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 40.0) = 1$ then
        return 40
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 50.0) = 1$ then
        return 50
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 60.0) = 1$ then
        return 60
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 70.0) = 1$ then
        return 70
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 80.0) = 1$ then
        return 80
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 90.0) = 1$ then
        return 90
    end if
    if $compareResourceParameters(deviceInformation, functionBlockInformation, parameter1, parameter2, 100.0) = 1$ then
        return 100
    end if
```

**Algorithm 9** Evaluate Memory Method

```
procedure EVALUATEMEMORY($deviceInformation$, $functionBlockInformation$, $parameter1$, $parameter2$)
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 0) = -1$ then
        return 100
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 0) = 0$ then
        return -25
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.00) = 1$ then
        return 0
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.10) = 1$ then
        return 10
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.20) = 1$ then
        return 20
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.30) = 1$ then
        return 30
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.40) = 1$ then
        return 40
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.50) = 1$ then
        return 50
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.60) = 1$ then
        return 60
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.70) = 1$ then
        return 70
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.80) = 1$ then
        return 80
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 1.90) = 1$ then
        return 90
    end if
    if $compareMemoryResources(deviceInformation, functionBlockInformation, parameter1, parameter2, 2.00) = 1$ then
        return 100
    end if
```

# Chapter 5

# Experiments and Results

In this chapter, the experiment's tests and results are presented and discussed, including the description of the pipeline utilized during the tests.

## 5.1 Pipeline Description

One crucial aspect to consider before the initial development and testing steps was how the system, namely the pipeline used for testing the efficacy and accuracy of the offloading mechanism, was defined and configured. The term pipeline, in this case, was considered to be the chaining of Function Blocks in a sequential manner, in such a way that Function Blocks, triggered either by external events or inputs, were stored in a queue list in a First-In-First-Out approach, and were therefore executed according to that list's order. Pipelines could be categorized into two groups:

- **Series pipelines** when Function Blocks execute one at a time, meaning that even if there are resources available, there is only progress when a Function Blocks finishes its execution.
- **Parallel pipelines** when Function Blocks execute concurrently, meaning that if there are resources available, it is possible to progress in the pipeline's execution by having devices run their Function Blocks at the same time.

Function Blocks needed to be mapped to devices independently of how the pipelines were configured. Devices could be considered homogeneous or heterogeneous, depending on how similar they were to one another in terms of specifications. Still, there are different advantages and disadvantages in choosing a system with either homogeneous or heterogeneous devices, depending on how the pipeline is categorized.

**For series pipelines**:

- **With Homogeneous Devices**: it is not possible to optimize bottlenecks, but at the same time, it is possible to minimize networking time by aggregating multiple blocks in a single device.

- **With Heterogeneous Devices**: it is also possible to minimize networking time by aggregating multiple blocks in a single device, and additionally, it is possible to optimize bottlenecks.

**For parallel pipelines**:

- **With Homogeneous Devices**: it is not possible to optimize bottlenecks, but at the same time, it is possible to minimize networking time but only to the point where the number of devices doesn't exceed the total parallel fluxes.
- **With Heterogeneous Devices**: it is possible to optimize bottlenecks.

Other restrictions to the pipeline included the possibility of specific Function Blocks being allocated to fixed and specific devices. Certain groups of Function Blocks needed to be assigned to the same device so that the pipeline didn't break.

With these constraints in mind, the pipeline was thought out, and it was planned as being in series with heterogeneous devices.

Presented from figures 5.1 to 5.5, the pipeline in which the algorithms would be tested was divided into two parts, the training pipeline, and the testing pipeline. Both pipeline lines began by loading a spreadsheet with data, training data for the first line of execution, and testing data for the second execution line. The steps of the training pipeline's execution, presented in figures 5.1 and 5.2 were:

1. **Loading of spreadsheet**: The test dataset was loaded onto the pipeline so that operations could be performed on it.
2. **Data grouping**: The data was grouped for analysis.
3. **Feature extraction**: Corresponds to the process of erasing irrelevant and redundant features from the dataset, improving accuracy when assigning text into one or more categories [36].
4. **Normalization**: Is the process of casting the dataset to a specific range. [30].
5. **Pickle normalization**: Similar to the previous step, but only explicitly done to pickles. Pickles are serialized or de-serialized Python object structures.
6. **PCA transformation**: Principal Component Analysis, or PCA, is the process of linear transformation applied from correlated variables to pairwise uncorrelated variables. [6]
7. **Labelling**: The data returned from the previous step was labeled.
8. **Save on spreadsheet**: The data was saved on a spreadsheet.

In this training pipeline, there were also other steps occurring concurrently after the PCA transformation step, as presented in figure 5.3, namely:

1. **Saving the PCA transformation result as a pickle**.

2. **Elliptic envelope**: This algorithm models the data as a high dimensional Gaussian distribution with possible co-variances between feature dimensions. In simpler terms, it attempts to find a boundary ellipse containing most of the data. [21]

3. **Saving the result of the elliptic envelope as a pickle**.

The testing pipeline, presented in figures 5.4 and 5.5 was very similar to the training pipeline, so only the steps that differed from the ones in the latter will be explained. Its execution order was as follows:

1. **Loading of spreadsheet**: The testing dataset was loaded onto the pipeline so that operations could be performed on it.

2. **Saving of spreadsheet**: The testing data set was initially saved on a separate spreadsheet before any operation was done on it.

3. **Event accumulator**: The event accumulator served as a composer to the pipeline in the sense that it waited until both previous steps were done before continuing execution.

4. **Data grouping**.

5. **Feature extraction**.

6. **Loading normalized pickle**: The pickle that was normalized on the training pipeline's execution was loaded.

7. **Normalization**.

8. **Pickle normalization**: similar to the previous step, but only explicitly done to pickles.

9. **Loading PCA transformed pickle**: The PCA transformation's result pickle, created in the training pipeline, was loaded.

10. **PCA transformation**.

11. **Loading pickle**: The pickle resulting from the final step of the training pipeline was loaded so that data could be used for the rest of the testing pipeline's execution.

12. **Labelling**.

13. **Elliptic envelope**.

14. **Save on spreadsheet**: The data resulting from the previous step was saved on a spreadsheet.

Since devices were spread through the network, the pipelines also included MQTT communication Function Blocks to pass data between them, necessary for the pipeline's good functioning. MQTT is an open Organization for the Advancement of Structured Information Standards

(OASIS) and an International Organization for Standardization (ISO) standard for the client-server messaging transport protocol. Based on a publish/subscribe communication pattern, the MQTT protocol presents characteristics such as its openness, simpleness, ease of deployment. Lightweight, the protocol is capable of transmitting data over low-bandwidth or unreliable networks with very low power consumption. It is constituted by a publisher, a subscriber, and a broker, also known as the MQTT server. The publisher is responsible for sending messages to a messaging service on a topic, while the subscriber receives all the messages sent to the channel whose topic it subscribed to. [29]

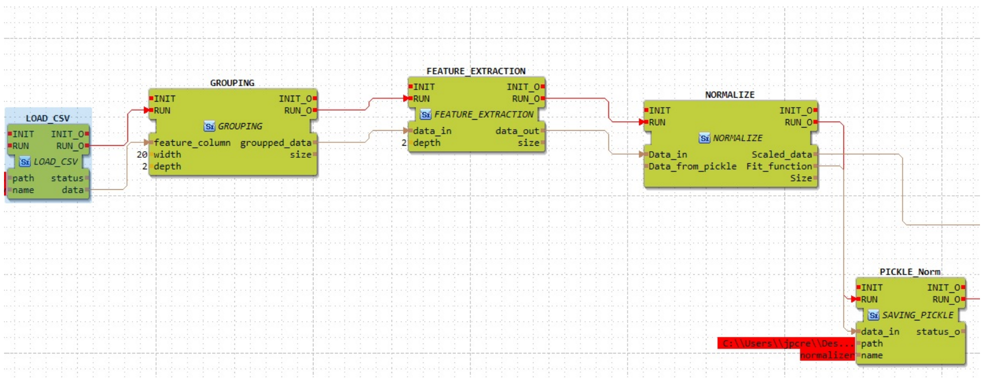


Figure 5.1: First Pipeline Part in Training Pipeline

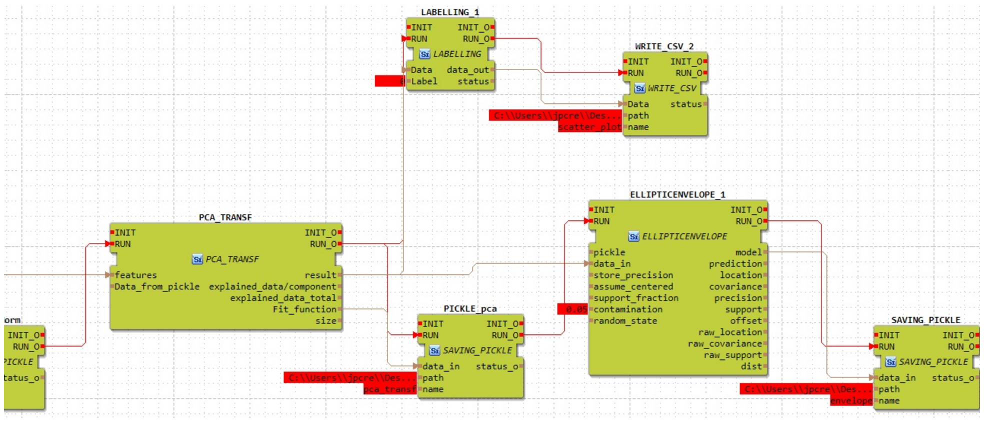


Figure 5.2: Second Pipeline Part in Training Pipeline

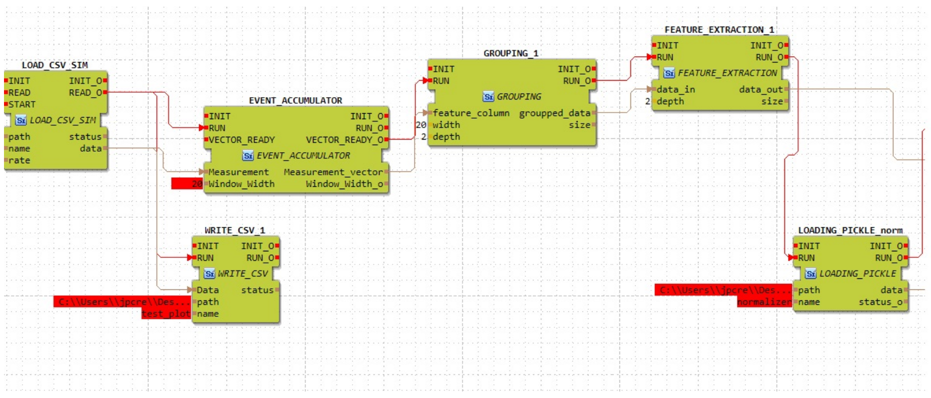


Figure 5.3: First Pipeline Part in Testing Pipeline

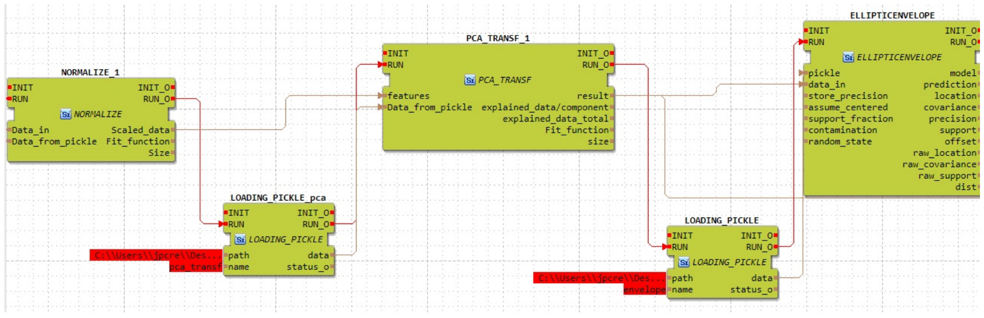


Figure 5.4: Second Pipeline Part in Testing Pipeline

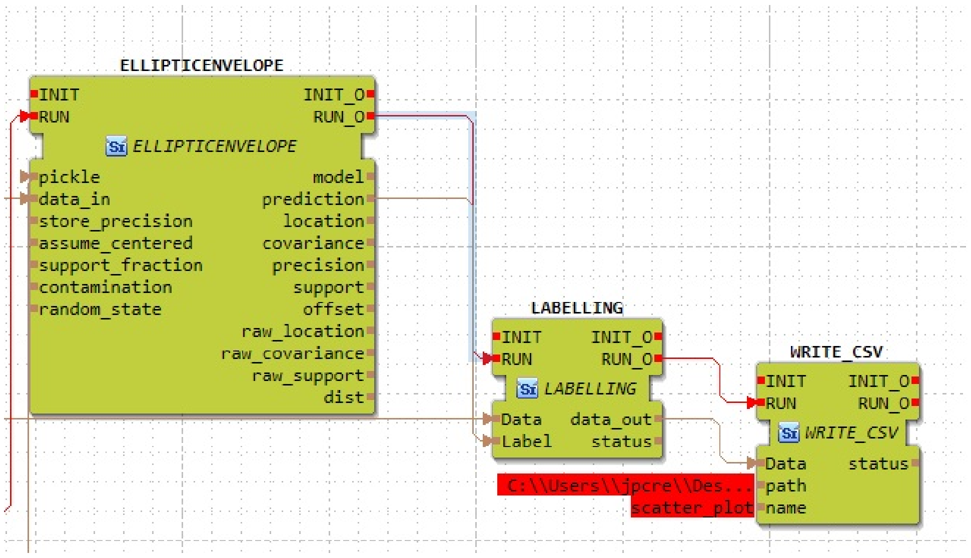
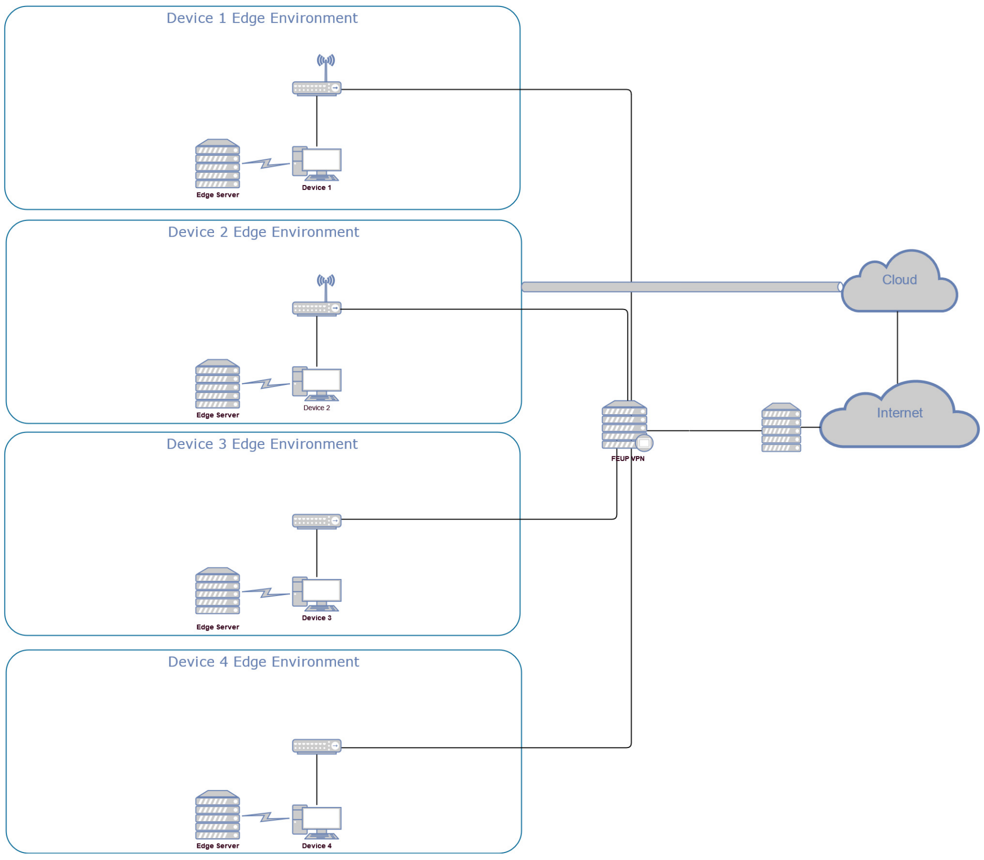


Figure 5.5: Third Pipeline Part in Testing Pipeline


Device 1 Edge Environment
Edge Server
Device 1
Device 2 Edge Environment
Edge Server
Device 2
Device 3 Edge Environment
Edge Server
Device 3
Device 4 Edge Environment
Edge Server
Device 4
FEUP VPN
Cloud
Internet
Figure 5.6: Network Topology of the Mechanism

## 5.2 Experiment Design

A couple of important points need to be presented regarding the testing process used to determine how effective and accurate the proposed algorithms were. Given the choice to make the mechanism interact with DINASORE more statically, not allowing for dynamic reconfiguration of Function Block to device mapping, to execute the tests, it was necessary to execute the pipeline before running the algorithms. But before the Function Blocks that composed the pipeline were allocated to their respective devices, the mechanism was executed to communicate with each device's DINASORE OPC-UA server instance and to perform the single parameter reading mentioned in subsection 4.2.1. To provide further input to the algorithm, the pipeline was executed for approximately 2 minutes per execution, after device to Function Block mapping was done. The configuration of the tests and results were described in a spreadsheet to provide a suggested optimization solution as an output. A critical consideration in testing was that different devices were used. Two of them were personal laptops, and the rest were owned by the DIGI-2 lab, which configured the devices for the experiment. The devices were listed and categorized as follows:

1. **Device 1**: 64-bit Windows 10 ASUS brand computer named DESKTOP-T6OSSL0. Featured an Intel(R) Core(TM) i7-4510U CPU with a core clock of 2.00 Ghz and featured 4 Gigabytes (GBs) of Random Access Memory (RAM).

2. **Device 2**: 64-bit Windows 10 Lenovo brand computer named LAPTOP-6DPA64B0. Featured an AMD Ryzen 7 5800H CPU with a core clock of 3.20 GHz and 16 GBs of RAM.

3. **Device 3**: 64-bit Intel brand mini-computer named DESKTOP-E2G2PKE. Featured an Intel(R) Core(TM) i3-7100U CPU with a core clock of 2.40 GHz and 4 GBs of RAM.

4. **Device 4**: 64-bit Windows 10 computer named DESKTOP-B005INA. Featured an Intel(R) Core(TM) i7 CPU with a core clock of 3.60 GHz and 8 GBs of RAM.

The manner in which the devices communicated between themselves, to constitute the mechanism's' network topology, is displayed in figure 5.6.

For Function Block distribution was considered as local, all the executions that were done on only one device, meaning that one device could handle all of the pipeline without utilizing the MQTT protocol. Nonetheless, the most important experiments were achieved by distributing different Function Blocks between other devices across the network, to understand the importance of having systems with various devices functioning simultaneously, and especially to understand if there were advantages in utilizing a Cloud solution in the mechanism. For some executions, the chosen device to be considered as the Cloud solution was device 2 since it was the device that possessed the best metrics among all the available devices.

Many executions were done to test all the algorithms with different input data values, and the results were all recorded in the previously mentioned spreadsheet. This spreadsheet had many subsheets, and a scenario of execution was represented for each. The different scenarios of execution scenarios were:

- **Local Scenario A**: local executions done on device 1.
- **Local Scenario B**: local executions done on device 2.
- **Local Scenario C**: local executions done on device 3.
- **Local Scenario D**: local executions done on device 4.
- **Distributed Scenario E**: distributed executions done on devices 1 and 2 (not considered Cloud level).
- **Distributed Scenario F**: distributed executions done on devices 2 (considered as Cloud level) and 3.
- **Distributed Scenario G**: distributed executions done on devices 2 (considered as Cloud level) and 4.

The spreadsheet's sub-sheets were divided, column-wise, by the test number, an identifier consisting of the scenario of execution scenario and the number of execution, for example, A - 1.3, which meant that the results were obtained from the third test performed on data that was generated at the first execution of the pipeline on device 1. The devices involved column is self-explanatory, while the pipeline column provided information on how the pipeline was attributed to the devices. The algorithm selected column presented the algorithm executed on that specific test, while the parameters column explained the input given to the mechanism so that the algorithm could perform. Finally, the metrics evaluated column explained which metrics the test was based on, and the results column presented a picture of the output obtained. The spreadsheet is available on a public link available in section 8.1, and the results will be presented in subsections by their respective scenario.

## 5.3 Results

### 5.3.1 Local Scenario A

For scenario A executions, the Greedy and Improved Greedy algorithms, obtained null results, since those two algorithms performed comparisons between devices to get a possible optimization solution. As displayed in table 5.2, the output would always return 0. Nonetheless, for the Q-Learning algorithm, different results were obtained to show the other Function Blocks' load provoked on the first device. Analyzing the results obtained, it was possible to observe that Function Blocks such as the grouping FBs and the elliptic envelope FBs, were the most computationally demanding to the first device. In contrast, some of the Function Blocks, such as the pickle and spreadsheet operations FBs displayed a consistent evaluation throughout the battery of tests. There were also a few Function Blocks that could be categorized as being in-between consistency and provoking a heavy load on the device. Namely, the event accumulator FB, which managed to improve its first score obtained in the second and third executions, but if on the fourth try the evaluation came close to the first one obtained, the final execution displayed a significantly worse score than previously detected. The labeling and normalization FBs could also be included in this category since their scores showed significant deviation between executions.

Table 5.1: Greedy and Improved Greedy results for Local Scenario A executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 1 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 1 | 2 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 1 | 3 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 1 | 4 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 2 | 1 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 2 | 2 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 2 | 3 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 2 | 4 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 3 | 1 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 3 | 2 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 3 | 3 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 3 | 4 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 4 | 1 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 4 | 2 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 4 | 3 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 4 | 4 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 5 | 1 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 5 | 2 | All FBs on Device 1 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 5 | 3 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 5 | 4 | All FBs on Device 1 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |

Table 5.2: Q-Learning results for Local Scenario A executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 5 | All FBs on Device 1 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction Score = 10515.22 Loading Normalized Pickle Score = 10671.17 Elliptic Envelope (Testing) Score = 10169.48 Write CSV (2nd Testing) Score = 11216.26 Event Accumulator Score = 10105.38 Loading Pickle Score = 11807.72 Normalize Score = 12026.64 Grouping Score = 9587.94 Second Normalize Score = 10889.96 Load CSV Score = 10066.68 Loading PCA Pickle = 10388.600 Saving Pickle Score = 13988.21 Write CSV 1 Score= 10091.98 Load CSV Sim Score = 10057.64 Elliptic Envelope 1(Training) Score = 9307.11 Grouping Score = 10256.34 Feature Extraction 1 Score = 11837.22 Labelling (Testing) Score = 10872.66 |

Table 5.2: Q-Learning results for Local Scenario A executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 2 | 5 | All FBs on Device 1 | Q-Learning | List Of Devices | CPU and memory resources | Pickle PCA Score = 11438.23 Feature Extraction 1 (Testing) Score = 10639.55 Loading Normalized Pickle Score = 10131.58 Elliptic Envelope (Testing) Score = 10719.38 Write CSV (2nd Testing) Score = 11796.26 Event Accumulator Score = 11739.45 Grouping (Training) Score = 9365.63 Grouping 1 (Testing) Score = 11905.35 TIRAR Normalize 1 (Testing) Score = 10889.96 Load (Training) CSV Score = 10066.68 PCA Pickle = 11438.27 Saving Pickle Score = 13998.15 Write CSV 1 (1st Testing) Score= 11965.43 Write CSV 2 (Training) Score= 8966.94 Load CSV Sim Score = 10222.59 Elliptic Envelope 1 (Training) Score = 11519.42 Grouping Score = 9365.63 PCA Transfer 1 (Testing) = 11163.66 Labelling (Testing) Score = 10872.66 Pickle Norm Score = 10243.01 |
| 3 | 5 | All FBs on Device 1 | Q-Learning | List Of Devices | CPU and memory resources | Normalize (Training) Score = 11068.25 Pickle PCA Score = 12024.94 Feature Extraction 1 (Testing) Score = 10639.55 Elliptic Envelope (Testing) Score = 10465.97 Write CSV (2nd Testing) Score = 10945.57 Grouping 1 (Testing) Score = 11291.34 Normalize 1 (Testing) Score = 9589.89 Write CSV 1 (1st Testing) Score= 11942.11 Load CSV Sim Score = 10835.35 Elliptic Envelope 1 (Training) Score = 11021.11 Grouping Score = 9365.63 PCA Transfer 1 (Testing) = 11163.66 Feature Extraction 1 (Testing) Score = 11348.33 Labelling (Testing) Score = 12188.34 Labelling 1 (Training) Score = 13998.25 Pickle Norm Score = 10243.01 |
| 4 | 5 | All FBs on Device 1 | Q-Learning | List Of Devices | CPU and memory resources | Write CSV 2 Score = 12622.85 Normalize 1 Score = 10237.46 Labelling Score = 10167.79 Grouping 1 Score = 10269.41 Loading Normalized Pickle Score = 10401.83 Pickle Normalization Score = 11295.15 Feature Extraction Score = 11888.86 PCA Transformation = 11383.10 Normalize Score = 11670.27 Feature Extraction 1 Score = 13998.17 Event Accumulator Score = 10452.88 Elliptic Envelope 1 Score = 10522.41 Load CSV Sim Score = 9847.70 Write CSV 1 Score = 11536.67 Write CSV Score = 10650.51 Elliptic Envelope Score = 11976.31 Loading Pickle Socre = 10583.36 |
| 5 | 5 | All FBs on Device 1 | Q-Learning | List Of Devices | CPU and memory resources | Event Accumulator Score = 9810.95 Loading Normalized Pickle = 11194.66 Pickle Normalization Score = 12366.79 Pickle PCA Score = 13998.26 PCA Transformation Score = 11750.82 Write CSV 2 Score = 11605.01 Grouping 1 Score = 11596.29 Normalize Score = 11040.84 Feature Extraction Score = 10725.97 Feature Extraction 1 Score = 10194.97 Load CSV Sim Score = 11123.85 Write CSV Score = 10303.57 Labelling Score = 10126.54 Elliptic Envelope 1 Score = 10798.35 Normalize 1 Score = 11257.90 Write CSV Score = 10772.10 Elliptic Envelope Score = 10167.74 |

### 5.3.2 Local Scenario B

For type B executions, the Greedy and Improved Greedy algorithms displayed null results as well, while for the Q-Learning algorithm, different results demonstrated the different Function Blocks' load provoked on the second device. Analyzing the results obtained in table 5.4, it was possible to observe that for different executions some Function Blocks revealed to be heavier independently of execution. Function Blocks such as the elliptic envelope FBs, particularly the one from the training pipeline, the normalize FBs, the labelling FBs, the write to spreadsheet file FBs, and the feature extraction FBs were amongst the most demanding Function Blocks for the second device. Other Function Blocks showed, on average, more consistent results, and could be considered easier to execute, resource wise. In this case, the PCA transfer FBs, the grouping FBs, both the load spreadsheet FBs, the pickle operations FBs displayed similar results between executions, and albeit not being the best, these results showed that they were less taxing to the second device. The least taxing Function Block for device 2 was the event accumulator Function Block, since on average it was the Function Block that scored the best result in all optimization solution suggestions. Overall, the second device was more capable of executing all of the Function Blocks than the first one, which was to be expected given that it is computationally more powerful.

Table 5.3: Greedy and Improved Greedy results for Local Scenario B executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 1 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 1 | 2 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 1 | 3 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 1 | 4 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 2 | 1 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 2 | 2 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 2 | 3 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 2 | 4 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 3 | 1 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 3 | 2 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 3 | 3 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 3 | 4 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 4 | 1 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 4 | 2 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 4 | 3 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 4 | 4 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 5 | 1 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 5 | 2 | All FBs on Device 2 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 5 | 3 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 5 | 4 | All FBs on Device 2 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |

Table 5.4: Q-Learning results for Local Scenario B executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 5 | All FBs on Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Write CSV Score = 18764.81 Saving Pickle Score = 17516.66 PCA Transformation 1 Score = 18844.35 Elliptic Envelope 1 Score = 14931.17 Labelling 1 Score = 15395.67 Feature Extraction 1 Score = 199784.48 Write CSV 1 Score = 19717.88 Normalize Score = 16924.79 Event Accumulator Score = 20083.95 Feature Extraction Score = 19817.56 Labelling Score = 17338.49 Write CSV 2 Score = 16235.48 Elliptic Envelope Score = 23996.95 Pickle PCA Score = 19053.07 Loading PCA Pickle Score = 18720.15 |
| 2 | 5 | All FBs on Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Loading Pickle Score = 15838.96 Pickle PCA Score = 19893.95 Load CSV Score = 15109.66 Feature Extraction Score = 17048.57 Write CSV Score = 20846.90 Pickle Normalization = 17001.52 Grouping Score = 18111.42 Write CSV 2 Score = 18642.55 PCA Trasnformation 1 Score = 19332.16 Grouping 1 Score = 17967.24 Saving Pickle Score = 17926.72 Write CSV 1 Score = 19043.13 Elliptic Envelope Score = 18465.80 Normalize 1 Score = 15484.90 Normalize Score = 18306.51 Labelling Score = 16614.37 |

Table 5.4: Q-Learning results for Local Scenario B executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 3 | 5 | All FBs on Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Loading PCA Pickle Score = 18971.04 Loading Normalized Pickle Score = 18323.69 PCA Transformation Score = 18906.51 Elliptic Envelope 1 Score = 17244.98 Write CSV Score = 19344.81 Labelling 1 Score = 19385.38 Feature Extraction 1 Score = 16648.42 Write CSV 2 Score = 16665.47 Elliptic Envelope Score = 18488.02 Saving Pickle Score = 16552.88 Feature Extraction Score = 16061.02 Grouping 1 Score = 17185.86 Grouping Score = 18549.34 Event Accumulator Score = 17273.07 Normalize 1 Score = 18701.80 Load CSV Score = 16837.02 Labelling Score = 23995.84 Write CSV 1 Score = 16694.04 PCA Transformation 1 Score = 16839.34 Normalize Score = 16567.78 Load CSV Sim Score = 19875.67 |
| 4 | 5 | All FBs on Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Saving Pickle Score = 18343.22 Normalize Score = 16595.48 Grouping Score = 18264.11 Loading Normalized Pickle Score = 15846.76 Write CSV 1 Score = 19322.83 Grouping 1 Score = 15728.94 Normalized Pickle Score = 20218.62 Elliptic Envelope 1 Score = 21585.62 Feature Extraction 1 Score = 17763.86 Write CSV Score = 16177.64 Pickle PCA Score = 15398.38 PCA Transformation Score = 17905.78 Labelling Score = 14471.56 PCA Transformation 1 Score = 23996.85 Normalize 1 Score = 17128.09 Load CSV Score = 17212.08 Event Accumulator Score = 18097.01 Loading PCA Pickle Score = 19616.98 Load CSV Sim Score = 17797.82 |
| 5 | 5 | All FBs on Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Event Accumlator Score = 14662.55 Grouping Score = 17684.79 Labelling 1 Score = 17434.03 Feature Extraction Score = 16906.44 Write CSV 2 Score = 16627.09 Loading Normalized Pickle Score = 19090.78 Loading Pickle Score = 19562.76 Loading Pickle PCA Score = 17436.56 Normalize 1 Score = 18155.07 Feature Extraction Score = 15512.10 Pickle PCA Score = 18963.46 Grouping 1 Score = 23996.91 PCA Transformation 1 Score = 17040.40 Load CSV Score = 17053.48 Elliptic Envelope Score = 18575.41 Normalize Score = 18908.28 |

### 5.3.3 Local Scenario C

The Greedy and Improved Greedy algorithms still displayed null results for scenario C executions. In contrast, the Q-Learning algorithm again demonstrated the different Function Blocks' load provoked on the third device. Analyzing the results obtained in table 5.6, interesting results were obtained, considering that on average, the algorithm's evaluation of Function Block needs was as positive, and in some cases, even better, than the one obtained for the second device. This could be explained by the fact that device 2 was utilized in the deployment and control of the pipelines with the 4DIAC-IDE and therefore needed to allocate some resources to perform those functions. But if the differences between the second and third devices weren't too extreme, the differences between the first and third devices were worthy of mention. By observing the results obtained in both cases, it was easily verified that the Q-Learning evaluation results were almost double for most of the Function Blocks that were executed. The fact that device 1 was older and had little available space in its memory could have influenced this. The second device's readings showed that it had more memory and CPU available for Function Block execution compared to those same metrics on the first device. In terms of the Q-Learning's Function Block evaluation, the results obtained indicated that for the third device, the Function Block toll was very inconsistent, given that many Function Blocks had results almost opposite from one execution to the next. Examples of this behavior were the PCA transformation FBs, the PCA pickle FB loading, the normalized pickle FB, the normalized FB, the grouping FBs, and the write to CSV (Comma-Separated Values) files FBs. Other Function Blocks displayed consistent results, such as the loading of the CSV files FBs, the feature extraction FBs, and the saving pickle FB. They could be considered as the ones that, on average, weighted less to the device's resources.

Table 5.5: Greedy and Improved Greedy results for Local Scenario C executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 1 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 1 | 2 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 1 | 3 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 1 | 4 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 2 | 1 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 2 | 2 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 2 | 3 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 2 | 4 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 3 | 1 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 3 | 2 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 3 | 3 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 3 | 4 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 4 | 1 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 4 | 2 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 4 | 3 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 4 | 4 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 5 | 1 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 5 | 2 | All FBs on Device 3 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 5 | 3 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 5 | 4 | All FBs on Device 3 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |

Table 5.6: Q-Learning results for Local Scenario C executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 5 | All FBs on Device 3 | Q-Learning | List Of Devices | CPU and memory resources | Loading Pickle PCA Score = 19340.47 Loading Normalized Pickle = 17760.29 PCA Transformation 1 Score = 19894.91 Normalize 1 Score = 16308.94 Write CSV 1 Score = 18262.15 Event Accumulator Score = 17441.66 Loading Pickle Score = 20032.18 Groupin 1 Score = 19095.19 Normalized Pickle Score = 17721.37 Feature Extraction 1 Score = 19753.60 Labelling 1 Score = 22997.16 Write CSV Score = 17697.45 Grouping Score = 18669.40 Elliptic Envelope 1 Score = 19493.41 Load CSV Sim Score = 15655.74 Elliptic Envelope Score = 17940.00 |
| 2 | 5 | All FBs on Device 3 | Q-Learning | List Of Devices | CPU and memory resources | Loading Pickle Score = 15838.96 Pickle PCA Score = 19893.95 Load CSV Score = 15109.66 Feature Extraction Score = 17048.57 Write CSV Score = 20846.90 Pickle Normalization = 17001.52 Grouping Score = 18111.42 Write CSV 2 Score = 18642.55 PCA Trasnformation 1 Score = 19332.16 Grouping 1 Score = 17967.24 Saving Pickle Score = 17926.72 Write CSV 1 Score = 19043.13 Elliptic Envelope Score = 18465.80 Normalize 1 Score = 15484.90 Normalize Score = 18306.51 Labelling Score = 16614.37 |

Table 5.6: Q-Learning results for Local Scenario C executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 3 | 5 | All FBs on Device 3 | Q-Learning | List Of Devices | CPU and memory resources | PCA Transformation 1 Score = 14623.11 Loading Normalized Pickle Score = 15394.25 Load CSV Score = 16623.14 Grouping Score = 14848.59 Elliptic Envelope Score = 16987.08 Pickle PCA Score = 15996.64 Write CSV Score = 164141.81 Elliptic Envelope 1 Score = 15352.93 Write CSV 2 Score = 17218.57 Saving Pickle Score = 17222.38 Grouping 1 Score = 16771.04 Loading Pickle PCA Score = 17595.41 Normalize Score = 21997.22 Labelling Score = 14305.63 Labelling 1 Score = 16407.91 PCA Transformation Score = 14164.21 Load CSV Sim Score = 16262.93 Event Accumulator Score = 11858.75 Write CSV 1 Score = 15350.02 Feature Extraction 1 Score = 18769.82 Loading Pickle Score = 16466.90 |
| 4 | 5 | All FBs on Device 3 | Q-Learning | List Of Devices | CPU and memory resources | Loading Pickle Score = 18636.15 Elliptic Envelope 1 Score = 19389.15 Normalize 1 Score = 19812.04 PCA Transformation 1 Score = 18426.55 Normalize Score = 14949.44 Write CSV 2 Score = 19614.41 Labelling 1 Score = 16615.44 Write CSV 1 Score = 20012.07 Grouping Score = 23996.98 Feature Extraction 1 Score = 18458.00 Loading Normalized Pickle Score = 19409.912 Load CSV Sim Score = 18048.00 Event Accumulator Score = 17923.27 Grouping 1 Score = 17343.90 Loading Pickle PCA Score = 17455.35 Write CSV Score = 16961.35 Elliptic Envelope Score = 18845.37 |
| 5 | 5 | All FBs on Device 3 | Q-Learning | List Of Devices | CPU and memory resources | Saving Pickle Score = 18843.91 Normalize Score = 17950.07 Feature Extraction Score = 18588.04 Loading Pickle PCA Score = 18862.77 Pickle PCA Score = 17700.83 Write CSV 1 Score = 17958.02 Feature Extraction 1 Score = 19234.01 Pickle Normalization Score = 23997.03 PCA Transformation 1 Score = 18130.47 Write CSV Score = 19139.50 Grouping 1 Score = 20271.99 Loading Normalized Pickle Score = 18283.03 Load CSV Sim Score = 19754.00 Event Accumulator Score = 16337.94 Grouping Score = 19408.81 Elliptic Envelope 1 Score = 16899.25 Loading Pickle Score = 17880.47 Normalize 1 Score = 16026.25 Elliptic Envelope Score = 17065.42 |

### 5.3.4 Local Scenario D

For the final individual testing, scenario D executions' Greedy and Improved Greedy algorithms results didn't change from the previous executions. As for the Q-Learning algorithm results, as seen in table 5.8, displayed a consistent behavior in the sense that it wasn't possible to identify a definite and clear group of best and worst-performing Function Blocks. On many executions, Function Blocks obtained sequential evaluations of their best and worst scores while obtaining on other executions evaluation values that oscillated between these peaks of scores.

Table 5.7: Greedy and Improved Greedy results for for Local Scenario D executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 1 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 1 | 2 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 1 | 3 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 1 | 4 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 2 | 1 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 2 | 2 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 2 | 3 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 2 | 4 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 3 | 1 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 3 | 2 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |

Table 5.7: Greedy and Improved Greedy results for for Local Scenario D executions

| Execution Number | Test Number | Pipeline | Algorithm Selected | Parameters Tested | Metrics Evaluated | Results |
|---|---|---|---|---|---|---|
| 3 | 3 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 3 | 4 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 4 | 1 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 4 | 2 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 4 | 3 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 4 | 4 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |
| 5 | 1 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | CPU resources | Device Score = 0 |
| 5 | 2 | All FBs on Device 4 | Greedy Algorithm | List Of Devices | Memory resources | Device Score = 0 |
| 5 | 3 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | Device Score = 0 |
| 5 | 4 | All FBs on Device 4 | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | Device Score = 0 |

Table 5.8: Q-Learning results for for Local Scenario D executions

| Execution Number | Test Number | Pipeline | Algorithm Selected | Parameters Tested | Metrics Evaluated | Results |
|---|---|---|---|---|---|---|
| 1 | 5 | All FBs on Device 4 | Q-Learning | List Of Devices | CPU and memory resources | Loading Normalized Pickle Socre = 20249.35 Grouping Score = 17231.00 Write CSV Score = 21569.28 Event Accumulator Score = 19659.62 Saving Pickle Score = 26996.45 Labelling 1 Score = 21325.94 Labelling Score = 19329.78 Load CSV Score = 18844.62 Normalize Score = 18273.80 Feature Extraction 1 Score = 17660.58 PCA Transformation 1 Score = 16406.90 Write CSV 1 Score = 18458.77 Elliptic Envelope 1 Score = 22089.46 Pickle PCA Score = 17503.57 Write CSV 2 Score = 19917.57 Loading Pickle Score = 19559.83 Load CSV Sim Score = 20944.91 Normalize 1 Score = 20482.39 Loading Pickle PCA Score = 15610.08 |
| 2 | 5 | All FBs on Device 4 | Q-Learning | List Of Devices | CPU and memory resources | Labelling Score = 22444.01 Loading Pickle Score = 20472.70 Write CSV 2 Score = 18709.24 PCA Transformation 1 Score = 17370.46 Pickle Norm Score = 22124.74 Pickle PCA Score = 19228,97 Elliptic Envelope 1 Score = 20410.71 Loading Normalized Pickle Score = 18704.91 Saving Pickle Score = 21748.20 Feature Extraction 1 Score = 18540.80 Grouping Score = 20925.59 Write CSV 1 Score = 15157.50 PCA Transformation Score = 20359.38 Elliptic Envelope Score = 19009.92 Labelling 1 Score = 20998.97 Feature Extraction Score = 19275.21 Event Accumulator Score = 21460.10 Load CSV Sim Score = 19357.42 Load CSV Score = 22180.95 Grouping 1 Score = 19746.18 Loading Pickle PCA Score = 26996.55 Write CSV Score = 19484.36 |
| 3 | 5 | All FBs on Device 4 | Q-Learning | List Of Devices | CPU and memory resources | Loading Pickle Score = 15594.07 Elliptic Envelope Score = 20317.66 Loading Normalized Pickle Score = 19880.31 Grouping Score = 16619.10 Write CSV 1 Score = 21345.84 Labelling Score = 17066.30 Pickle Normalization Score = 20748.07 Feature Extraction Score = 19585.26 PCA Transformation 1 Score = 22322.87 Saving Pickle Score = 21234.09 Labelling 1 Score = 18837.82 Load CSV Score = 21086.00 Loading Pickle PCA Score = 21491.05 Event Accumulator Score = 26996.56 Load CSV Sim Score = 19059.36 Grouping Score = 20789.66 Elliptic Envelope Score = 21078 Normalize 1 Score = 20374.51 |
| 4 | 5 | All FBs on Device 4 | Q-Learning | List Of Devices | CPU and memory resources | Event Accumulator Score = 20592.98 Loading Pickle PCA Score = 20762.83 Feature Extraction Score = 17601.90 Normalize Score = 18599.02 Loading Pickle Score = 20750.94 Elliptic Envelope 1 Score = 19968.49 Load CSV Score = 19298.52 Labelling Score = 19329.44 Write CSV 1 Score = 21528.17 PCA Transformation Score = 21034.93 Grouping 1 Score = 19719.01 Normalize 1 Score = 21427.70 Saving Pickle Score = 16829.47 Labelling 1 Score = 20746.67 |

Table 5.8: Q-Learning results for for Local Scenario D executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 5 | 5 | All FBs on Device 4 | Q-Learning | List Of Devices | CPU and memory resources | Load CSV Score = 19662.21 Loading Normalizeds Pickle Score = 16118,96 Write CSV 2 Score = 18191.25 Grouping Score = 18643.33 Saving Pickle Score = 26996.66 PCA Transformation Score = 21272.18 Labelling 1 Score = 17458.69 Feature Extraction 1 Score = 18159.30 Grouping 1 Score = 19809.71 Elliptic Envelope Score = 18508.42 Event Accumulator Score = 18060.19 Write CSV Score = 18743.28 Pickle Normalization Score = 17511.89 Loading Pickle PCA Score = 22080.74 Feature Extraction Score = 20834.40 Labelling Score = 21789.00 Write CSV 1 Score = 22139.97 Load CSV Sim Score = 20858.20 PCA Transformation 1 Score = 18934.97 Normalize 1 Score = 13273.53 Loading Pickle Score = 21727.76 |

### 5.3.5 Distributed Scenario E

For scenario E to scenario G executions, the Greedy and Improved Greedy offered actual results, given that these executions included two devices to run the whole pipeline. As previously stated, an MQTT protocol broker was utilized in these distributed cases to pass data from one device to another. In these cases, the Q-Learning algorithm demonstrated for each device all of the Function Block load evaluation, meaning that even if a device hadn't executed a specific Function Block, it would still assume the resources necessary to execute those Function Blocks in the device. Analyzing the results obtained, that visible in table 5.10, and considering only the Greedy and Improved Greedy algorithms first, it was unsurprisingly verified that for all executions, the second device fared better when compared to the first device, on both memory and CPU metrics. For the Q-Learning algorithm, the second device presented, once again, for the majority of the Function Blocks, better scores when compared to the first device. Curiously, the results obtained individually for each device showed somewhat identical conclusions to the ones observed in subsections 5.3.1 and 5.3.2, respectively. For the first device, the elliptic envelope, and grouping FBs, more often than not, needed more resources than the other FBs. For the second device, the previously mentioned grouping FB needed plenty of resources, in addition to the elliptic envelope FB, which was very taxing on the it. At the same time, it was also possible to verify that the labeling FBs, the pickle operation FBs, and the feature extraction FBs, once again revealed to need many resources for their execution. An interesting behavior was verified in the grouping FB, given that for each execution, the results for these devices were opposite. For example, the grouping FB obtained an incredibly high evaluation on the first device in the first execution while obtaining a very low score for the second one. Still, on the second execution, the roles were inverted, where the scores were almost perfectly opposite of what had been obtained on the first execution, meaning the first device with a low score and the second device with a high score. This behavior happened on three of the five executions, while on one those five executions the results obtained between both devices were closer to one another. The event accumulator FB was a Function Block whose performance on both devices consistently displayed the smallest difference. At the same time, the rest of the pickle operations and spreadsheet operations also verified minor differences between device evaluations. Finally, the MQTT publisher FB had a surprisingly higher score on the first device, but this could be explained by the fact that the MQTT broker was hosted on the second device and therefore helped in spending more resources outside of the scope of the Function Block execution. The MQTT subscriber FB displayed higher results on the first device. Still, it couldn't be reallocated to that device, given that device 1 had no conditions to host the MQTT broker.

Table 5.9: Greedy and Improved Greedy results for Distributed Scenario E executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 1 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 1 | 2 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 1 | 3 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-T6OSSL0 Score = -22 |
| 1 | 4 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-T6OSSL0 Score = -22 |
| 2 | 1 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 2 | 2 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 2 | 3 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-T6OSSL0 Score = -22 |
| 2 | 4 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-T6OSSL0 Score = -22 |
| 3 | 1 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 3 | 2 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 3 | 3 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = 20 DESKTOP-T6OSSL0 Score = -20 |
| 3 | 4 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 20 DESKTOP-T6OSSL0 Score = -20 |
| 4 | 1 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |

Table 5.9: Greedy and Improved Greedy results for Distributed Scenario E executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 4 | 2 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 4 | 3 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = 21 DESKTOP-T6OSSL0 Score = -21 |
| 4 | 4 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 21 DESKTOP-T6OSSL0 Score = -21 |
| 5 | 1 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 5 | 2 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-T6OSSL0 Score = -1 |
| 5 | 3 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-T6OSSL0 Score = -22 |
| 5 | 4 | Device 1 - Half of FBs and Device 2 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-T6OSSL0 Score = -22 |

Table 5.10: Q-Learning results for Distributed Scenario E executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction Device 1 Score = 7420.43 Normalize 1 Device 1 Score = 10157.44 Loading Normalized Pickle Device 1 Score = 11617.78 Write CSV Device 1 Score = 7194.09 Normalize Device 1 Score = 22738.20 Pickle Norm Device 1 Score = 12540.48 Event Accumulator Device 1 Score = 8037.15 Feature Extraction 1 Device 1 Score = 12788.85 Grouping 1 Device 1 Score = 9972.73oad CSV Sim Device 1 Score = 5497.80 MQTT Publisher Device 1 Score = 5490.23 MQTT Publisher 1 Device 1 Score = 12729.26 Pickle PCA Device 1 Score = 5134.40 Loading Pickle Device 1 Score = 7570.38 Loading Pickle PCA Device 1 Score = 7397.31 MQTT Subscriber Device 1 Score = 9810.16 MQTT Subscriber 1 Device 1 Score = 10135.32 Elliptic Envelope 1 Device 1 Score = 13226.92 Labelling Device 1 Score = 10047.74 PCA Transform 1 Device 1 Score = 7316.96 Elliptic Envelope Device 1 Score = 6877.81 Write CSV Device 1 Score = 8181.73 |
| 1 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction Device 2 Score = 9396.63 Normalize 1 Device 2 Score = 13638.16 Loading Normalized Pickle Device 2 Score = 11105.84 Write CSV Device 2 Score = 7974.76 Normalize Device 2 Score = 12310.59 Pickle Norm Device 2 Score = 8644.25 Event Accumulator Device 2 Score = 10970.00 Feature Extraction 1 Device 2 Score = 12506.65 Grouping 1 Device 2 Score = 26692.97 Load CSV Sim Device 2 Score = 9015.45 MQTT Publisher Device 2 Score = 9861.09 MQTT Publisher 1 Device 2 Score = 11327.33 Pickle PCA Device 2 Score = 6149.15 Loading Pickle Device 2 Score = 6452.54 Loading Pickle PCA Device 2 Score = 10729.24 MQTT Subsriber Device 2 Score = 8359.91 MQTT Subscriber 1 Device 2 Score = 11343.70 Elliptic Envelope 1 Device 2 Score = 6000.85 Labelling Device 2 Score = 11655.97 PCA Transform 1 Device 2 Score = 12807.58 Elliptic Envelope Device 2 Score = 9918.21 Write CSV Device 2 Score = 9340.08 |

Table 5.10: Q-Learning results for Distributed Scenario E executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 2 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction Device 1 Score = 10263.13 Normalize 1 Device 1 Score = 7926.96 Loading Normalized Pickle Device 1 Score = 8415.65 Write CSV 1 Device 1 Score = 6522.07 Normalize Device 1 Score = 13342.77 Pickle Norm Device 1 Score = 10768.97 Event Accumulator Device 1 Score = 12219.34 Feature Extraction 1 Device 1 Score = 7990.06 Grouping 1 Device 1 Score = 22727.45 Load CSV Sim Device 1 Score = 8826.46 MQTT Publisher Device 1 Score = 8642.47 MQTT Publisher 1 Device 1 Score = 7397.12 Pickle PCA Device 1 Score = 13981.65 Loading Pickle Device 1 Score = 11824.76 Loading Pickle PCA Device 1 Score = 9351.70 MQTT Subscriber Device 1 Score = 9650.58 MQTT Subscriber 1 Device 1 Score = 11919.09 Elliptic Envelope 1 Device 1 Score = 6004.56 Labelling Device 1 Score = 6140.01 PCA Transform 1 Device 1 Score = 9940.75 Elliptic Envelope Device 1 Score = 7316.66 Write CSV Device 1 Score = 8675.63 |
| 2 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction Device 2 Score = 13358.38 Normalize 1 Device 2 Score = 12097.38 Loading Normalized Pickle Device 2 Score = 7214.97 Write CSV 1 Device 2 Score = 8542.04 Normalize Device 2 Score = 26691.34 Pickle Norm Device 2 Score = 9680.04 Event Accumulator Device 2 Score = 12356.23 Feature Extraction 1 Device 2 Score = 10506.11 Grouping 1 Device 2 Score = 5649.83 Load CSV Sim Device 2 Score = 13560.69 MQTT Publisher Device 2 Score = 7007.76 MQTT Publisher 1 Device 2 Score = 8754.60 Pickle PCA Device 2 Score = 10611.30 Loading Pickle Device 2 Score = 11687.11 Loading Pickle Device 2 Score = 11824.76 MQTT Subscriber Device 2 Score = 10474.76 MQTT Subscriber 1 Device 2 Score = 6497.24 Elliptic Envelope 1 Device 2 Score = 11432.44 Labelling Device 2 Score = 6862.80 PCA Transform 1 Device 2 Score = 10924.28 Elliptic Envelope Device 2 Score = 12053.95 Write CSV Device 2 Score = 10044.352 |
| 3 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction 1 Device 1 Score = 5766.05 Load CSV Device 1 Score = 9459.01 Feature Extraction Device 1 Score = 7126.21 Grouping Device 1 Score = 18788.58 Pickle Normalization Device 1 Score = 5965.90 Event Accumulator Device 1 Score = 4503.08 Loading Normalized Pickle Device 1 Score = 6473.22 Normalize Device 1 Score = 11042.07 Write CSV Device 1 Score = 3814.92 Load CSV Sim Device 1 Score = 6481.23 Loading Pickle Device 1 Score = 15346.28 Pickle PCA Device 1 Score = 5770.32 Labelling Device 1 Score = 9201.10 MQTT Subscriber Device 1 Score = 7123.39 MQTT Subscriber Device 1 Score = 7180.20 Write CSV Device 1 Score = 6026.70 Elliptic Envelope Device 1 Score = 8590.82 PCA Transformation 1 Device 1 Score = 9656.54 Loading Pickle PCA Device 1 Score = 4865.60 |
| 3 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction 1 Device 2 Score = 10230.57 Load CSV Device 2 Score = 7831.64 Feature Extraction Device 2 Score = 10393.13 Grouping Device 2 Score = 6426.32 Grouping 1 Device 2 Score = 8670.94 Pickle Normalization Device 2 Score = 11899.08 Event Accumulator Device 2 Score = 26692.57 Loading Normalized Pickle Device 2 Score = 12810.14 Normalize Device 2 Score = 12634.01 Write CSV Device 2 Score = 5499.36 Load CSV Sim Device 2 Score = 16188.16 Loading Pickle Device 2 Score = 10215.56 Pickle PCA Device 2 Score = 6132.39 Labelling Device 2 Score = 6983.48 MQTT Subscriber Device 2 Score = 13108.00 MQTT Subscriber 1 Device 2 Score = 12122.58 Write CSV Device 2 Score = 10647.67 Elliptic Envelope Device 2 Score = 10569.29 PCA Transformation 1 Device 2 Score = 17137.84 Loading Pickle PCA Device 2 Score = 10928.62 |
| 4 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Load CSV Device 1 Score = 6370.57 Feature Extraction Device 1 Score = 5298.82 Loading Normalized Pickle Device 1 Score = 4960.36 Normalize 1 Device 1 Score = 7831.65 Write CSV 1 Device 1 Score = 7756.46 Grouping Device 1 Score = 10488.30 Pickle Normalization Device 1 Score = 10181.85 Normalize Device 1 Score = 10143.10 Load CSV Sim Device 1 Score = 7407.57 Event Accumulator Device 1 Score = 6952.77 Feature Extraction 1 Device 1 Score = 6663.45 Labelling 1 Device 1 Score = 10233.19 Loading Pickle PCA Device 1 Score = 8650.55 Loading Pickle Device 1 Score = 18782.89 Labelling Device 1 Score = 9384.36 MQTT Subscriber Device 1 Score = 5456.38 MQTT Subscriber 1 Device 1 Score = 7689.92 Elliptic Envelope 1 Device 1 Score = 9496.28 Write CSV Device 1 Score = 7798.32 Elliptic Envelope Device 1 Score = 6743.80 PCA Transformation 1 Device 1 Score = 6936.63 |

Table 5.10: Q-Learning results for Distributed Scenario E executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 4 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Load CSV Device 2 Score = 10539.91 Feature Extraction Device 2 Score = 148441.22 Loading Normalized Pickle Device 2 Score = 12177.62 Normalize 1 Device 2 Score = 9897.88 Write CSV 1 Device 2 Score = 11804.02 Grouping Device 2 Score = 26691.65 Pickle Normalization Device 2 Score = 10622.91 Normalize Device 2 Score = 12423.49 Load CSV Sim Device 2 Score = 7322.38 Event Accumulator Device 2 Score = 8085.62 Feature Extraction 1 Device 2 Score = 7122.95 Labelling 1 Device 2 Score = 7682.22 Loading Pickle PCA Device 2 Score = 12559.91 Loading Pickle Device 2 Score = 9817.36 Labelling Device 2 Score = 8187.54 MQTT Subscriber Device 2 Score = 15661.73 MQTT Subscriber 1 Device 2 Score = 11547.37 Elliptic Envelope 1 Device 2 Score = 5575.86 Write CSV Device 2 Score = 6736.40 Elliptic Envelope Device 2 Score = 13167.76 PCA Transformation 1 Device 2 Score = 16579.15 |
| 5 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction 1 Device 1 Score = 7385.42 Write CSV 1 Device 1 Score = 7042.70 Grouping 1 Device 1 Score = 5878.39 Grouping Device 1 Score = 7952.64 Normalize Device 1 Score = 9806.67 Pickle Normalization Device 1 Score = 5094.00 Loading Normalized Pickle Device 1 Score = 17789.43 Load CSV Device 1 Score = 6024.77 Load CSV Sim Device 1 Score = 9553.02 Event Accumulator Device 1 Score = 9188.79 MQTT Publisher 1 Device 1 Score = 5442.91 Labelling 1 Device 1 Score = 5930.00 Saving Pickle Device 1 Score = 4795.09 Pickle PCA Device 1 Score = 9422.57 Labelling Device 1 Score = 7831.11 MQTT Subscriber 1 Device 1 Score = 2644.13 MQTT Subscriber Device 1 Score = 6829.71 Elliptic Envelope 1 Device 1 Score = 8218.61 Loading Pickle PCA Device 1 Score = 8794.33 Write CSV Device 1 Score = 7689.92 Elliptic Envelope Device 1 Score = 6916.97 Loading Pickle Device 1 Score = 7510.91 |
| 5 | 5 | Device 1 - Half of FBs and Device 2 | Q-Learning | List Of Devices | CPU and memory resources | Feature Extraction 1 Device 2 Score = 6677.92 Write CSV 1 Device 2 Score = 6258.20 Grouping 1 Device 2 Score = 15918.84 Grouping Device 2 Score = 26692.69 Normalize Device 2 Score = 111067.46 Pickle Normalization Device 2 Score = 11678.01 Loading Normalized Pickle Device 2 Score = 13582.61 Load CSV Device 2 Score = 9033.56 Load CSV Sim Device 2 Score = 8161.94 Event Accumulator Device 2 Score = 6848.10 MQTT Publisher 1 Device 2 Score = 5293.87 Labelling 1 Device 2 Score = 6254.37 Saving Pickle Device 2 Score = 7309.80 Pickle PCA Device 2 Score = 13096.62 Labelling Device 2 Score = 5822.55 MQTT Subscriber 1 Device 2 Score = 11501.63 MQTT Subscriber Device 2 Score = 11498.54 Elliptic Envelope 1 Device 2 Score = 16165.12 Loading Pickle PCA Device 2 Score = 12195.94 Write CSV Device 2 Score = 6567.59 Elliptic Envelope Device 2 Score = 12028.02 Loading Pickle Device 2 Score = 6071.50 |

### 5.3.6 Distributed Scenario F

For scenario F, the Greedy and Improved Greedy algorithm showed curious results, as for all the executions, the second device scored better in terms of memory metrics, but the third device consistently scored better in terms of CPU metrics. As stated previously, the fact that device 2 was utilized as the machine where the 4DIAC-IDE was executed and controlled and where the MQTT broker was functioning could have influenced the level of CPU usage on it, therefore making it worse when compared to the device 3. As for the Q-Learning algorithm, this battery of tests considered an associated cost of using the second device of 0.5. That meant that the value obtained would be reduced in half for every reward calculated after evaluating the CPU and memory resources. Given this information and analyzing table 5.12, it was possible to understand just how much cost could be a decisive factor in evaluating the adequacy of executing a Function Block on a given device, since, for all executions, the third device found that for a majority of Function Blocks much higher evaluation had been obtained, when compared to the second device. These results were only possible due to the associated cost since individual executions of scenarios

A and C demonstrated that the values should have been much closer to those obtained in scenario F. Interestingly, on F scenario executions, the evaluation values for the third device were, on average, much smaller than when compared to the executions of scenario C. In this case, the time device 3 spent waiting for MQTT messages in the channel it subscribed to could explain this discrepancy.

Table 5.11: Greedy and Improved Greedy results for Distributed Scenario F executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 1 | Device 2 - Half of FBs and Device 3 - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = -1 DESKTOP-E2G2PKE Score = 1 |
| 1 | 2 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-E2G2PKE Score = -1 |
| 1 | 3 | Device 2 - Half of FBs and Device 3 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -22 DESKTOP-E2G2PKE Score = 22 |
| 1 | 4 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-E2G2PKE Score = -22 |
| 2 | 1 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = -1 DESKTOP-E2G2PKE Score = 1 |
| 2 | 2 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-E2G2PKE Score = -1 |
| 2 | 3 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -24 DESKTOP-E2G2PKE Score = 24 |
| 2 | 4 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 24 DESKTOP-E2G2PKE Score = -24 |
| 3 | 1 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LLAPTOP-6DPA64B0 Score = -1 DESKTOP-E2G2PKE Score = 1 |
| 3 | 2 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-E2G2PKE Score = -1 |
| 3 | 3 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -24 DESKTOP-E2G2PKE Score = 24 |
| 3 | 4 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 24 DESKTOP-E2G2PKE Score = -24 |

Table 5.11: Greedy and Improved Greedy results for Distributed Scenario F executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 4 | 1 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = -1 DESKTOP-E2G2PKE Score = 1 |
| 4 | 2 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-E2G2PKE Score = -1 |
| 4 | 3 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -24 DESKTOP-E2G2PKE Score = 24 |
| 4 | 4 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 24 DESKTOP-E2G2PKE Score = -24 |
| 5 | 1 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = -1 DESKTOP-E2G2PKE Score = 1 |
| 5 | 2 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-E2G2PKE Score = -1 |
| 5 | 3 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -18 DESKTOP-E2G2PKE Score = 18 |
| 5 | 4 | Device 2 - Half of FBs and Device 3 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 18 DESKTOP-E2G2PKE Score = -18 |

Table 5.12: Q-Learning results for Distributed Scenario F executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | PCA Transformation Device 2 Score = 5552.44 Labelling 1 Device 2 Score = 6829.26 Write CSV 2 Device 2 Score = 4012.59 Elliptic Envelope Device 2 Score = 4741.79 Write CSV Device 2 Score = 6042.72 Labelling Device 2 Score = 4979.86 MQTT Subscriber Device 2 Score = 4130.65 MQTT Subscriber 1 Device 2 Score = 4231.56 Elliptic Envelope 1 Device 2 Score = 2663.96 PCA Transformation 1 Device 2 Score = 4833.05 Loading Pickle PCA Device 2 Score = 3206.99 PCA Transformation Device 2 Score = 12854.82 Labelling 1 Device 2 Score = 5363.14 Write CSV 2 Device 2 Score = 4100.01 Elliptic Envelope Device 2 Score = 6498.20 Write CSV Device 2 Score = 3663.28 Labelling Device 2 Score = 4131.46 MQTT Subscriber Device 2 Score = 3545.64 MQTT Subscriber 1 Device 2 Score = 5071.87 Elliptic Envelope Device 2 Score = 5009.58 PCA Transformation 1 Device 2 Score = 4478.02 Loading Pickle PCA Device 2 Score = 5788.20 |

Table 5.12: Q-Learning results for Distributed Scenario F executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 1 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | PCA Transformation Device 3 Score = 9992.80 Labelling 1 Device 3 Score = 9563.84 Write CSV 2 Device 3 Score = 4327.25 Elliptic Envelope Device 3 Score = 7792.98 Write CSV Device 3 Score = 9614.76 Labelling Device 3 Score = 11327.04 MQTT Subscriber Device 3 Score = 12260.05 MQTT Subscriber 1 Device 3 Score = 11939.52 Elliptic Envelope 1 Device 3 Score = 5017.12 PCA Transformation 1 Device 3 Score = 5059.99 Loading Pickle PCA Device 3 Score = 7300.80 PCA Transformation Device 3 Score = 23730.00 Labelling 1 Device 3 Score = 9193.16 Write CSV 2 Device 3 Score = 10697.68 Elliptic Envelope Device 3 Score = 11232.67 Write CSV Device 3 Score = 6535.36 Labelling Device 3 Score = 12564.38 MQTT Subscriber Device 3 Score = 10892.38 MQTT Subscriber 1 Device 3 Score = 10985.12 Elliptic Envelope Device 3 Score = 12456.25 PCA Transformation 1 Device 3 Score = 5875.23 Loading Pickle PCA Device 3 Score = 1007.43 |
| 2 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | PCA Transformation 1 Device 2 Score = 5215.87 PCA Transformation 1 Device 3 Score = 5314.04 Write CSV Device 2 Score = 3131.04 Write CSV Device 3 Score = 3951.55 Write CSV 2 Device 2 Score = 4326.61 Write CSV 2 Device 3 Score = 10743.11 Labelling 1 Device 2 Score = 5410.14 Labelling 1 Device 3 Score = 9552.97 Pickle PCA Device 2 Score = 12854.08 Pickle PCA Device 3 Score = 22734.42 MQTT Subscriber 1 Device 2 Score = 3560.32 MQTT Subscriber 1 Device 3 Score = 8267.80 MQTT Subscriber Device 2 Score = 3553.64 MQTT Subscriber Device 3 Score = 9315.13 Elliptic Envelope 1 Device 2 Score = 4392.05 Elliptic Envelope 1 Device 3 Score = 8268.81 Elliptic Envelope Device 2 Score = 6899.70 Elliptic Envelope Device 3 Score = 8880.34 Loading Pickle Device 2 Score = 4330.54 Loading Pickle Device 3 Score = 7789.38 Loading Pickle PCA Device 2 Score = 4254.66 Loading Pickle PCA Device 3 Score = 7443.39 Labelling Device 2 Score = 2936.16 Labelling Device 3 Score = 10354.35 PCA Transformation 1 Device 2 Score = 2224.55 PCA Transformation 1 Device 3 Score = 11413.18 Write CSV Device 2 Score = 3776.90 Write CSV Device 3 Score = 7518.63 Write CSV 2 Device 2 Score = 5490.97 Write CSV 2 Device 3 Score = 6713.92 Labelling 1 Device 2 Score = 5025.39 Labelling 1 Device 3 Score = 8310.47 Pickle PCA Device 2 Score = 6565.94 Pickle PCA Device 3 Score = 9876.92 MQTT Subscriber 1 Device 2 Score = 4741.81 MQTT Subscriber 1 Device 3 Score = 7764.64 MQTT Subscriber Device 2 Score = 3305.61 MQTT Subscriber Device 3 Score = 6389.86 Elliptic Envelope 1 Device 2 Score = 6328.61 Elliptic Envelope 1 Device 3 Score = 120903.90 Elliptic Envelope Device 2 Score = 3927.54 Elliptic Envelope Device 3 Score = 8431.96 Loading Pickle Device 2 Score = 3019.32 Loading Pickle Device 3 Score = 8396.80 Loading Pickle PCA Device 2 Score = 4944.66 Loading Pickle PCA Device 3 Score = 11404.00 Labelling Device 2 Score = 3225.06 Labelling Device 3 Score = 10409.40 |
| 2 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | PCA Transformation 1 Device 2 Score = 5215.87 Write CSV Device 2 Score = 3131.04 Write CSV 2 Device 2 Score = 4326.61 Labelling 1 Device 2 Score = 5410.14 Pickle PCA Device 2 Score = 12854.08 MQTT Subscriber 1 Device 2 Score = 3560.32 MQTT Subscriber Device 2 Score = 3553.64 Elliptic Envelope 1 Device 2 Score = 4392.05 Elliptic Envelope Device 2 Score = 6899.70 Loading Pickle Device 2 Score = 4330.54 Loading Pickle PCA Device 2 Score = 4254.66 Labelling Device 2 Score = 2936.16 PCA Transformation 1 Device 2 Score = 2224.55 Write CSV Device 2 Score = 3776.90 Write CSV 2 Device 2 Score = 5490.97 Labelling 1 Device 2 Score = 5025.39 Pickle PCA Device 2 Score = 6565.94 MQTT Subscriber 1 Device 2 Score = 4741.81 MQTT Subscriber Device 2 Score = 3305.61 Elliptic Envelope 1 Device 2 Score = 6328.61 Elliptic Envelope Device 2 Score = 3927.54 Loading Pickle Device 2 Score = 3019.32 Loading Pickle PCA Device 2 Score = 4944.66 Labelling Device 2 Score = 3225.06 |

Table 5.12: Q-Learning results for Distributed Scenario F executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 3 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | PCA Transformation 1 Device 3 Score = 7439.65 Write CSV Device 3 Score = 3840.06 Write CSV 2 Device 3 Score = 5932.23 Labelling 1 Device 3 Score = 9336.40 Pickle PCA Device 3 Score = 6610.69 MQTT Subscriber 1 Device 3 Score = 9450.11 MQTT Subscriber Device 3 Score = 7709.53 Elliptic Envelope 1 Device 3 Score = 6238.09 Elliptic Envelope Device 3 Score = 7222.85 Loading Pickle Device 3 Score = 8277.69 Loading Pickle PCA Device 3 Score = 7353.85 Labelling Device 3 Score = 7652.31 PCA Transformation 1 Device 3 Score = 8827.02 Write CSV Device 3 Score = 12161.51 Write CSV 2 Device 3 Score = 9269.36 Labelling 1 Device 3 Score = 23730.78 Pickle PCA Device 3 Score = 10182.36 MQTT Subscriber 1 Device 3 Score = 8336.52 MQTT Subscriber Device 3 Score = 9883.77 Elliptic Envelope 1 Device 3 Score = 6883.44 Elliptic Envelope Device 3 Score = 9652.81 Loading Pickle Device 3 Score = 10227.34 Loading Pickle PCA Device 3 Score = 7093.34 Labelling Device 3 Score = 11596.78 |
| 4 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | PCA Transformation 1 Device 2 Score = 5281.03 Write CSV Device 2 Score = 2717.64 Write CSV 2 Device 2 Score = 4027.44 Labelling 1 Device 2 Score = 12854.49 Pickle PCA Device 2 Score = 3108.15 MQTT Subscriber 1 Device 2 Score = 2831.52 MQTT Subscriber Device 2 Score = 5730.78 Elliptic Envelope 1 Device 2 Score = 5880.24 Elliptic Envelope Device 2 Score = 3734.45 Loading Pickle Device 2 Score = 3936.07 Loading Pickle PCA Device 2 Score = 2523.58 Labelling Device 2 Score = 4843.84 PCA Transformation 1 Device 2 Score = 3888.37 Write CSV Device 2 Score = 4721.02 Write CSV 2 Device 2 Score = 7916.83 Labelling 1 Device 2 Score = 4832.22 Pickle PCA Device 2 Score = 5504.82 MQTT Subscriber 1 Device 2 Score = 6383.88 MQTT Subscriber Device 2 Score = 5726.77 Elliptic Envelope 1 Device 2 Score = 6166.70 Elliptic Envelope Device 2 Score = 4171.84 Loading Pickle Device 2 Score = 1737.93 Loading Pickle PCA Device 2 Score = 5438.02 Labelling Device 2 Score = 4401.09 |
| 4 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | PCA Transformation 1 Device 3 Score = 9232.86 Write CSV Device 3 Score = 7968.59 Write CSV 2 Device 3 Score = 23726.87 Labelling 1 Device 3 Score = 6966.62 Pickle PCA Device 3 Score = 7423.60 MQTT Subscriber 1 Device 3 Score = 6845.85 MQTT Subscriber Device 3 Score = 6081.12 Elliptic Envelope 1 Device 3 Score = 6408.70 Elliptic Envelope Device 3 Score = 3186.57 Loading Pickle Device 3 Score = 6642.63 Loading Pickle PCA Device 3 Score = 12454.59 Labelling Device 3 Score = 6955.08 PCA Transformation 1 Device 3 Score = 11698.06 Write CSV Device 3 Score = 8673.28 Write CSV 2 Device 3 Score = 9756.91 Labelling 1 Device 3 Score = 8341.11 Pickle PCA Device 3 Score = 9858.15 MQTT Subscriber 1 Device 3 Score = 9543.05 MQTT Subscriber Device 3 Score = 10439.72 Elliptic Envelope 1 Device 3 Score = 6214.54 Elliptic Envelope Device 3 Score = 7738.00 Loading Pickle Device 3 Score = 8264.28 Loading Pickle PCA Device 3 Score = 6113.81 Labelling Device 3 Score = 9159.76 |
| 5 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | Labelling 1 Device 2 Score = 3283.30 Saving Pickle Device 2 Score = 5781.99 Write CSV 2 Device 2 Score = 12850.22 Write CSV Device 2 Score = 4784.11 Labelling Device 2 Score = 7344.29 MQTT Subscriber Device 2 Score = 5165.78 MQTT Subscriber 1 Device 2 Score = 5998.42 Elliptic Envelope 1 Device 2 Score = 5703.42 Elliptic Envelope Device 2 Score = 5424.27 Labelling 1 Device 2 Score = 7029.68 Saving Pickle Device 2 Score = 6458.62 Write CSV 2 Device 2 Score = 5379.03 Write CSV Device 2 Score = 4503.42 Labelling Device 2 Score = 5228.99 MQTT Subscriber Device 2 Score = 6478.61 MQTT Subscriber 1 Device 2 Score = 6501.49 Elliptic Envelope 1 Device 2 Score = 6598.40 Elliptic Envelope Device 2 Score = 4491.82 |
| 5 | 5 | Device 2 - Half of FBs and Device 3 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.5 cost) | Labelling 1 Device 3 Score = 12332.44 Saving Pickle Device 3 Score = 11204.09 Write CSV 2 Device 3 Score = 10205.88 Write CSV Device 3 Score = 12828,58 Labelling Device 3 Score = 8536.07 MQTT Subscriber Device 3 Score = 8748.06 MQTT Subscriber 1 Device 3 Score = 11513.70 Elliptic Envelope 1 Device 3 Score = 10625.71 Elliptic Envelope Device 3 Score = 95517.45 Labelling 1 Device 3 Score = 13560.81 Saving Pickle Device 3 Score = 11816.95 Write CSV 2 Device 3 Score = 23735.47 Write CSV Device 3 Score = 8964.68 Labelling Device 3 Score = 8780.34 MQTT Subscriber Device 3 Score = 9047.34 MQTT Subscriber 1 Device 3 Score = 12160.91 Elliptic Envelope 1 Device 3 Score = 10360.01 Elliptic Envelope Device 3 Score = 10282.81 |
| + | | | | | | |

### 5.3.7 Distributed Scenario G

For the last scenario of experiments, G scenario executions' results regarding both Greedy and Improved Greedy showed that for 4 out of 5 total executions, device 2 had better evaluations on memory resources. In comparison, device 4 had better evaluations on CPU resources. In all the tests, the one outlier was the final execution, where the second device scored better than the fourth one for both metrics. When it came to Q-Learning results, present in table 5.14, it was indeed verified that by decreasing the second device's associated offloading cost to 0.25, a generalized increase of evaluation scores, for device 2 was obtained since, individually, it displayed better results when compared to what was verified in F scenario executions. Nonetheless, and generally speaking, device 2 evaluation scores for all tests that were made still scored significantly worse when compared to device 4 evaluation scores. Few of the exceptions were related to pickle operations, especially the loading of the normalized pickle FB and occasionally the loading of the training pipeline's spreadsheet and the final write to the spreadsheet in the testing pipeline. Once again, taking the fourth device's scores, it was impossible to categorize FBs by their performance since evaluation values oscillated substantially between different executions. In terms of MQTT-related FBs, the findings present in subsection 5.3.6 still held throughout G scenario executions.

Table 5.13: Greedy and Improved Greedy results for Distributed Scenario G executions

| Execution Number | Test Number | Pipeline | Algorithm Selected | Parameters Tested | Metrics Evaluated | Results |
|---|---|---|---|---|---|---|
| 1 | 1 | Device 2 - Half of FBs and Device 4 - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-B005INA Score = -1 |
| 1 | 2 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-B005INA Score = -1 |
| 1 | 3 | Device 2 - Half of FBs and Device 4 - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-B005INA Score = -22 |
| 1 | 4 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-B005INA Score = -22 |
| 2 | 1 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = -1 DESKTOP-B005INA Score = 1 |
| 2 | 2 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-B005INA Score = -1 |
| 2 | 3 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -22 DESKTOP-B005INA Score = 22 |
| 2 | 4 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-B005INA Score = -22 |

Table 5.13: Greedy and Improved Greedy results for Distributed Scenario G executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 3 | 1 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LLAPTOP-6DPA64B0 Score = -1 DESKTOP-B005INA Score = 1 |
| 3 | 2 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-B005INA Score = -1 |
| 3 | 3 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -21 DESKTOP-B005INA Score = 21 |
| 3 | 4 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 21 DESKTOP-B005INA Score = -21 |
| 4 | 1 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = -1 DESKTOP-B005INA Score = 1 |
| 4 | 2 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-B005INA Score = -1 |
| 4 | 3 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = -22 DESKTOP-B005INA Score = 22 |
| 4 | 4 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 22 DESKTOP-B005INA Score = -22 |
| 5 | 1 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | CPU resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-B005INA Score = -1 |
| 5 | 2 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Greedy Algorithm | List Of Devices | Memory resources | LAPTOP-6DPA64B0 Score = 1 DESKTOP-B005INA Score = -1 |
| 5 | 3 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | CPU resources | LAPTOP-6DPA64B0 Score = 23 DESKTOP-B005INA Score = -23 |
| 5 | 4 | Device 2 - Half of FBs and Device 4 - Other Half of FBs - Other Half of FBs | Improved Greedy Algorithm | List Of Devices and List Of Function Blocks | Memory resources | LAPTOP-6DPA64B0 Score = 23 DESKTOP-B005INA Score = -23 |

Table 5.14: Q-Learning results for Distributed Scenario G executions

| Execution Number | Test Number | Pipeline | Algorithm Selected | Parameters Tested | Metrics Evaluated | Results |
|---|---|---|---|---|---|---|
| 1 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 2 Score = 6111.14 Normalize Device 2 Score = 7272.28 Grouping 1 Device 2 Score = 7547.66 Load CSV Device 2 Score = 20020.82 Write CSV 1 Device 2 Score = 6969.55 Pickle Normalization Device 2 Score = 9532.04 Feature Extraction 1 Device 2 Score = 8735.14 Event Accumulator Device 2 Score = 7360.62 Normalize 1 Device 2 Score = 8698.97 Load CSV Sim Device 2 = 7814.65 Write CSV 2 Device 2 Score = 5808.25 Pickle PCA Device 2 = 10520.04 Elliptic Envelope Device 2 Score = 6786.19 Labelling 1 Device 2 Score = 3643.04 Loading Pickle PCA Device 2 Score = 8555.35 Loading Pickle Device 2 Score = 6061.18 PCA Transformation 1 Device 2 Score = 8602.16 Saving Pickle Device 2 Score = 6239.79 Labelling Device 2 Score = 5784.57 MQTT Subscriber 1 Device 2 Score = 5937.57 MQTT Subscriber Device 2 Score = 7793.85 Write CSV Device 2 Score = 7955.32 |
| 1 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 4 Score = 12315.26 Normalize Device 4 Score = 11016.02 Grouping 1 Device 4 Score = 8779.68 Load CSV Device 4 Score = 27678.34 Write CSV 1 Device 4 Score = 12463.72 Pickle Normalization Device 4 Score = 13168.75 Feature Extraction 1 Device 4 Score = 5316.92 Event Accumulator Device 4 Score = 15145.17 Normalize 1 Device 4 Score = 11139.78 Load CSV Sim Device 4 = 10219.65 Write CSV 2 Device 4 Score = 9069.86 Pickle PCA Device 4 = 10197.12 Elliptic Envelope Device 4 Score = 16725.18 Labelling 1 Device 4 Score = 7138.80 Loading Pickle PCA Device 4 Score = 10427.98 Loading Pickle Device 4 Score = 11976.27 PCA Transformation 1 Device 4 Score = 9375.09 Saving Pickle Device 4 Score = 10704.99 Labelling Device 4 Score = 7881.29 MQTT Subscriber 1 Device 4 Score = 9140.20 MQTT Subscriber Device 4 Score = 13764.54 Write CSV Device 4 Score = 8305.79 |
| 2 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 2 Score = 8481.73 Feature Extraction Device 2 Score = 8959.91 Write CSV 1 Device 2 Score = 11746.87 Feature Extraction 1 Device 2 Score = 8234.89 Pickle Normalization Device 2 Score = 8770.63 Event Accumulator Device 2 Score = 82255.17 Grouping 1 Device 2 Score = 4232.36 Normalize 1 Device 2 Score = 12210.02 Loading Normalized Pickle Device 2 Score = 20021.56 Load CSV Sim Device 2 Score = 6821.62 MQTT Publisher 1 Device 2 Score = 8596.58 Loading Pickle PCA Device 2 Score = 8396.25 Elliptic Envelope Device 2 Score = 7784.56 Labelling Device 2 Score = 8319.11 Write CSV 2 Device 2 Score = 7172.28 Write CSV Device 2 Score = 6682.33 Pickle PCA Device 2 Score = 6079.00 Elliptic Envelope 1 Device 2 Score = 7062.40 PCA Transformation 1 Device 2 Score = 6833.23 Loading Pickle Device 2 Score = 6732.39 MQTT Subscriber 1 Device 2 Score = 7798.53 MQTT Subscriber Device 2 Score = 5771.00 MQTT Subscriber Device 4 Score = 12043.88 |
| 2 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 4 Score = 8612.79 Feature Extraction Device 4 Score = 16034.59 Write CSV 1 Device 4 Score = 14714.40 Feature Extraction 1 Device 4 Score = 8762.93 Pickle Normalization Device 4 Score = 10002.80 Event Accumulator Device 4 Score = 10419.14 Grouping 1 Device 4 Score = 11019.73 Normalize 1 Device 4 Score = 8835.81 Loading Normalized Pickle Device 4 Score = 15923.33 Load CSV Sim Device 4 Score = 8830.93 MQTT Publisher 1 Device 4 Score = 11706.49 Loading Pickle PCA Device 4 Score = 5648.58 Elliptic Envelope Device 4 Score = 9939.90 Labelling Device 4 Score = 12850.21 Write CSV 2 Device 4 Score = 4829.58 Write CSV Device 4 Score = 27682.78 Pickle PCA Device 4 Score = 9492.55 Elliptic Envelope 1 Device 4 Score = 10447.17 PCA Transformation 1 Device 4 Score = 9658.41 Loading Pickle Device 4 Score = 10659.68 MQTT Subscriber 1 Device 4 Score = 10500.69 MQTT Subscriber Device 4 Score = 12043.88 |
| 3 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 2 Score = 8560.53 Write CSV 1 Device 2 Score = 7998.11 Normalize Device 2 Score = 4688.61 Feature Extraction Device 2 Score = 8098.33 Loading Normalized Pickle Device 2 Score = 9984.53 Normalize 1 Device 2 Score = 9619.18 Pickle Normalization Device 2 Score = 9577.90 Event Accumulator Device 2 Score = 5265.76 Load CSV Sim Device 2 Score = 9862.73 MQTT Publisher 1 Device 2 Score = 6326.61 Labelling Device 2 Score = 20021.31 Pickle PCA Device 2 Score = 8232.24 PCA Transformation Device 2 Score = 8897.66 Write CSV 2 Device 2 Score = 9397.12 Write CSV Device 2 Score = 9505.34 PCA Transformation 1 Device 2 Score = 8526.11 Loading Pickle PCA Device 2 Score = 7833.38 Loading Pickle Device 2 Score = 7444.59 Elliptic Envelope Device 2 Score = 10432.99 MQTT Subscriber Device 2 Score = 6693.86 MQTT Subscriber 1 Device 2 Score = 10644.66 |

Table 5.14: Q-Learning results for Distributed Scenario G executions

| **Execution Number** | **Test Number** | **Pipeline** | **Algorithm Selected** | **Parameters Tested** | **Metrics Evaluated** | **Results** |
|---|---|---|---|---|---|---|
| 3 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 4 Score = 13143.27 Write CSV 1 Device 4 Score = 227690.16 Normalize Device 4 Score = 7447.28 Feature Extraction Device 4 Score = 13191.77 Loading Normalized Pickle Device 4 Score = 4891.61 Normalize 1 Device 4 Score = 7100.50 Pickle Normalization Device 4 Score = 12598.97 Event Accumulator Device 4 Score = 11722.93 Load CSV Sim Device 4 Score = 11768.75 MQTT Publisher 1 Device 4 Score = 10471.72 Labelling Device 4 Score = 14467.90 Pickle PCA Device 4 Score = 8548.27 PCA Transformation Device 4 Score = 12409.48 Write CSV Device 4 Score = 14171.16 PCA Transformation 1 Device 4 Score = 6349.11 Loading Pickle PCA Device 4 Score = 12765.31 Loading Pickle Device 4 Score = 13184.52 Elliptic Envelope Device 4 Score = 8774.43 MQTT Subscriber Device 4 Score = 9450.04 MQTT Subscriber 1 Device 4 Score = 9399.26 |
| 4 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Normalize Device 2 Score = 9673.72 Load CSV Device 2 Score = 7416.07 Feature Extraction Device 2 Score = 9881.70 Feature Extraction 1 Device 2 Score = 8024.57 Write CSV 1 Device 2 Score = 20018.00 Pickle Normalization Device 2 Score = 9228.15 Event Accumulator Device 2 Score = 8524.90 Grouping Device 2 Score = 7563.25 Load CSV Sim Device 2 Score = 9479.13 Loading Pickle Device 2 Score = 8666.74 Saving Pickle Device 2 Score = 6890.93 Pickle PCA Device 2 Score = 6866.46 PCA Transformation 1 Device 2 Score = 6858.21 Labelling Device 2 Score = 4665.89 Elliptic Envelope Device 2 Score = 7829.01 Write CSV 2 Device 2 Score = 7743.71 Write CSV Device 2 Score = 6593.58 Elliptic Envelope 1 Device 2 Score = 8018.09 Loading Pickle PCA Device 2 Score = 5514.12 Labelling 1 Device 2 Score = 3971.61 MQTT Subscriber Device 2 Score = 8288.43 MQTT Subscriber 1 Device 2 Score = 9015.07 |
| 4 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Normalize Device 4 Score = 10007.97 Load CSV Device 4 Score = 8956.54 Feature Extraction Device 4 Score = 6974.39 Feature Extraction 1 Device 4 Score = 14675.85 Write CSV 1 Device 4 Score = 10356.33 Pickle Normalization Device 4 Score = 13891.69 Event Accumulator Device 4 Score = 15054.66 Grouping Device 4 Score = 9306.03 Load CSV Sim Device 4 Score = 12867.58 Loading Pickle Device 4 Score = 11960.07 Saving Pickle Device 4 Score = 12066.92 Pickle PCA Device 4 Score = 10319.38 PCA Transformation 1 Device 4 Score = 7455.17 Labelling Device 4 Score = 10447.26 Elliptic Envelope Device 4 Score = 12213.72 Write CSV 2 Device 4 Score = 10148.21 Write CSV Device 4 Score = 14093.45 Elliptic Envelope 1 Device 4 Score = 15641.98 Loading Pickle PCA Device 4 Score = 14903.41 Labelling 1 Device 4 Score = 27674.40 MQTT Subscriber Device 4 Score = 9266.56 MQTT Subscriber 1 Device 4 Score = 11685.67 |
| 5 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 2 Score = 6167.25 Load CSV Device 2 Score = 5817.48 Write CSV 1 Device 2 Score = 20016.43 Grouping 1 Device 2 Score = 5696.77 Event Accumulator Device 2 Score = 11173.58 Feature Extraction 1 Device 2 Score = 8375.44 Normalize Device 2 Score = 6616.10 Pickle Normalization Device 2 Score = 9947.39 Load CSV Sim Device 2 Score = 8541.54 Loading Normalized Pickle Device 2 Score = 9781.92 Loading Pickle Device 2 Score = 7165.05 Loading Pickle PCA Device 2 Score = 6960.41 Elliptic Envelope 1 Device 2 Score = 5054.23 Saving Pickle Device 2 Score = 4623.90 Write CSV Device 2 Score = 8822.76 Pickle PCA Device 2 Score = 7059.63 Elliptic Envelope Device 2 Score = 7946.72 Labelling Device 2 Score = 8378.995 Labelling 1 Device 2 Score = 6340.19 PCA Transformation 1 Device 2 Score = 7298.53 Write CSV 2 Device 2 Score = 8437.44 MQTT Subscriber 1 Device 2 Score = 6411.50 MQTT Subscriber Device 2 Score = 9241.48 |
| 5 | 5 | Device 2 - Half of FBs and Device 4 | Q-Learning | List Of Devices | CPU, memory resources and cost (device 2 0.25 cost) | Grouping Device 4 Score = 15076.02 Load CSV Device 4 Score = 10707.83 Write CSV 1 Device 4 Score = 4327.03 Grouping 1 Device 4 Score = 13201.84 Event Accumulator Device 4 Score = 9208.60 Feature Extraction 1 Device 4 Score = 10053.99 Normalize Device 4 Score = 8825.77 Pickle Normalization Device 4 Score = 8971.73 Load CSV Sim Device 4 Score = 11655.56 Loading Normalized Pickle Device 4 Score = 8966.13 Loading Pickle Device 4 Score = 5238.52 Loading Pickle PCA Device 4 Score = 10379.81 Elliptic Envelope 1 Device 4 Score = 13716.77 Saving Pickle Device 4 Score = 11512.67 Write CSV Device 4 Score = 14979.27 Pickle PCA Device 4 Score = 9721.49 Elliptic Envelope Device 4 Score = 9544.94 Labelling Device 4 Score = 5666.14 Labelling 1 Device 4 Score = 7877.75 PCA Transformation 1 Device 4 Score = 9790.50 Write CSV 2 Device 4 Score = 27682.67 MQTT Subscriber 1 Device 4 Score = 9430.07 MQTT Subscriber Device 4 Score = 17674.28 |

The findings obtained in the experiment allowed for a key set of ideas. The first was that while the task offloading mechanism developed offered three different optimization algorithms, the Greedy and its Improved Greedy variant worked in a straight-line manner. Given that it was easily understandable which devices could potentially score better in terms of CPU and memory resources, it can be said that the results offered by these algorithms didn't provide much detail in how substantial the difference between said devices was. This flaw could be considered somewhat corrected in the Q-Learning algorithm since it was possible to understand more clearly just how a Function Block and its individual needs could adapt to a device's available resources to offer. But it was only partially corrected given that, as shown, on many executions, the number of Function Blocks being evaluated and considered to be part of the system was never constant. This meant that while the mechanism could provide optimizations effectively, it could not accurately calculate Function Block needs. What justified this was how DINASORE stores the monitored Function Block information and overall resource monitoring, which made it, that on some executions, certain Function Blocks couldn't be attributed to any consumption since their initial and ending timestamps didn't match to any consumption timestamp, while also not being capable of being considered as zero consumption Function Blocks, given their initial and ending timestamps were different. In terms of consumption calculation, the Function Blocks the mechanism detected as more taxing were the grouping FB, the elliptic envelope FB, the normalize FB, and the labelling FB. Other examples of taxing Function Blocks, although not as consistently taxing as the ones mentioned previously were, the pickle normalization FB, the write to CSV FB, and the feature extraction FB, This was expected given that these Function Blocks were programmatically more complex than other Function Blocks. The final idea was related to the Cloud level and the associated cost of offloading parameter. Although the best device, in computational terms, device 2's evaluation in scenario F and G executions greatly suffered when the mechanism considered the cost in the reward attribution process. As expected, the more significant the cost, the higher the penalty, and even its resource superiority could not even the evaluations' scores as shown.

# Chapter 6

# Limitations and Future Work

In this chapter, the limitations resulting of the mechanism developed are presented, and future work to correct them is suggested. Criticism of the solution obtained is offered.

## 6.1 Limitations

When collecting information throughout the experiment, it was possible to identify limitations of the task offloading mechanism and, therefore, present some criticism of the development process. Firstly speaking about what was developed, a notable limitation of the mechanism was that the code present in the Main class had to be constantly altered to accommodate the different runs that were made to get results and obtain the device consumption information text files. More concisely, to get these files, it was necessary only to call ClientConnector class instances, while to obtain results, it was required only to call DataConnector and DeviceAvailabilityReader class instances, which meant that part of the Main class had to be rewritten whenever it was necessary to perform a measure of a devices' metrics, and when it was required to aggregate all the information available in text files to load consumption data on memory. Additionally, it was necessary to change the code in the main class for algorithm execution, as the Q-Learning output that was presented had a lot of information displayed to the user. Therefore it wasn't possible to run all the algorithms simultaneously without losing output information in the process. Other code alterations had to be done to correctly load data from different executions to the mechanism, although this was done on the Utils class. One other limitation explicitly related to the loading of data to the system was that on practically all algorithm tests, it was possible to verify that not all Function Blocks were evaluated, which compromised accuracy goals for the research results. Other limitations also existed, even if not wholly related to the code developed for the mechanism, but with constituents of the system with which the mechanism communicated. Namely, DINASORE's method for originating monitor and Function Block execution data, via text files, as previously explained, provoked this flaw in Function Block detection since there was no data available to attribute the intervals in which these undetected Function Blocks were performing their task, to timestamps of the monitoring of resources done by DINASORE. Another DINASORE related limitation consisted of the difficulty of attributing more than two devices for the execution of the whole pipeline since the implementation of the MQTT based Function Blocks didn't support sending and receiving all types of data, but only one and two-dimensional arrays, which limited the points in which the pipeline could be divided by different devices. Limitations related to devices themselves were rare, but the one that did exist was quite substantial since the device with better metrics, device 2, was the device utilized for initializing and controlling the pipeline's execution, which caused additional resource consumption on top of DINASORE and Function Block execution, and possibly skewed consumption data for said device. With this information, it is possible to state that some aspects of the research should be subjected to criticism, mainly, the manual process of gathering device resource information and loading consumption and device data onto the mechanism, which could be considered inconvenient since it at least doubled the number of times necessary to run the mechanism to get optimization results, the lack of accuracy in optimization results, as some Function Blocks consumption data was undetected by the mechanism and the lack of accurate device consumption information reading. Finally, the paucity of algorithm executions involving more than two devices should be a point of criticism, while Cloud level needed further exploration regarding security and latency factors that could be important in the choice of offloading a task.

## 6.2 Future Work

For all the limitations and respective criticism resulting from the development phase, future work could be done to overcome those aspects. Possible points to consider in future research and experiments include improvement in the algorithms developed. Simultaneously, adding new Reinforcement Learning related algorithms, such as Deep Q-Learning, Deep Deterministic Policy Gradient, for example, should be considered. Another point of improvement could be the integration between DINASORE and the task offloading mechanism. It would be important that the limitation regarding the undetected consumption of some Function Blocks' execution could be solved in following work. Although not part of the scope of the research, this could be achieved by modifying DINASORE's source code, mainly the part about the writing of the Function Block execution and resource parameter information, that could help in better detecting all of the Function Blocks deployed in the system. One possible solution that wouldn't require significant changes would be to reduce the interval of time in which the resources of the devices are monitored, making it more likely for the timestamps to be associated with the interval of Function Block execution. One possible change includes significant structural modifications to DINASORE and the task offloading mechanism. Mainly, how both components communicate makes it hard to monitor optimization solutions and consumption information in real-time, and simultaneously allow for dynamic pipeline reconfiguration. This last adjustment would eliminate the need to keep changing code on the Main class by allowing all of the information to be collected and data to be processed simultaneously, improving user experience with the mechanism. Considering DINASORE only, work should be done to ensure that a pipeline can be more easily divided between devices, independently of the type of data needed to send and receive from one Function Block to another. This would also aid in making algorithm executions with more device data, which could benefit in better understanding results and testing their efficiency more accurately. Finally, the Cloud level solution and its importance to task offloading mechanisms should be further studied and detailed. Other aspects should be strongly considered, such as security, since specific tasks may contain very sensitive information and therefore could be regarded as inadequate to send over the Internet, and latency, which in case of systems where the response times are critical could provide issues.

# Chapter 7

# Conclusions

In the initial chapters, questions and objectives relating to the research were defined. During the experiment, the accomplishment of the goals defined was attempted, since concurrently it would aid in answering those questions. Therefore, for the system at hand, the most critical variables in the environment that were considered were the CPU and memory resources, alongside possible offloading costs associated with a device. However, these choices were greatly influenced by the parameters monitored by DINASORE. In a system with different needs and monitoring methodology, the variables that affect the task offloading should reflect those aspects accordingly. It is possible to affirm that in terms of the design of the mechanism, that an abstract architecture was obtained. At the same time, it was also possible to realize that the bigger the abstraction of the communication between components, the larger the degree of the mechanism's abstraction can be. Some of DINASORE's defined architecture made it that the mechanism had to be more "rigid" than initially planned. In terms of algorithm conception, three algorithms were developed, with vastly different results and different levels of effectiveness. Nonetheless, no matter how the algorithms are planned out, they should account for the needs of the Function Blocks that are to be executed, for the available resources a device can provide for execution, specifically for all of the variables defined as offloading parameters. Although the algorithms offered expected results when understanding each of the devices involved metrics and overall Function Block consumption, it should be noted that it was hard to attribute a practical value to how efficient they were, since, at most, the number of devices involved in running the algorithms was two. Therefore the execution of algorithms with more complex device to Function Block mappings could provide better insight into the actual efficiency of what was developed. Regarding the importance of having a Cloud level, the research couldn't provide definitive evidence to support a positive or negative influence. What was clear was that having a Cloud level could prove crucial in cases where tasks are incredibly demanding, computationally speaking. Still, even then, the probable cost of acquiring a Cloud solution capable of efficiently running said tasks could prove to be prohibitive. Therefore one should carefully assess whether or not it is feasible and necessary to include Cloud solutions in task offloading mechanisms. In terms of improvements, it should be a priority to change how DINASORE communicates its consumption data to facilitate real-time optimization suggestions, dynamic pipeline reconfiguration, and user experience, followed by the improvement of existing algorithms and development of new ones. Finally, studying the integration of the Cloud level from other angles, such as security and latency problems, is encouraged.

# References

[1] Digital and Intelligent Industry Lab. http://systec-fof.fe.up.pt/systec/web_en.html. [Online; accessed 21-April-2021].

[2] Cambridge dictionary - production line. https://dictionary.cambridge.org/dictionary/english/production-line. [Online; accessed 23-April-2021].

[3] Mohammad Aazam, Sherali Zeadally, and Khaled A. Harras. Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems*, 87:278–289, October 2018.

[4] Khadija Akherfi, Micheal Gerndt, and Hamid Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, 14(1):1–16, January 2018.

[5] Dr Tarek M Attia. Challenges and Opportunities in the Future Applications of IoT Technology. *2nd Europe - Middle East - North African Regional Conference of the International Telecommunications Society (ITS): "Leveraging Technologies For Growth", Aswan, Egypt, 18th-21st February, 2019*, page 16.

[6] M. J. Baxter. Standardization and Transformation in Principal Component Analysis, with Applications to Archaeometry. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 44(4):513–527, 1995. Publisher: [Wiley, Royal Statistical Society].

[7] Eugen Borcoci. Edge and fog computing - convergence of solutions. *ComputationWorld 2018 Conference February 18, 2018, Barcelona*, page 118, 2018.

[8] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (IIoT): An analysis framework. *Computers in Industry*, 101:1–12, October 2018.

[9] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (IIoT): An analysis framework. *Computers in Industry*, 101:1–12, October 2018.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[11] Siguang Chen, Yimin Zheng, Kun Wang, and Weifeng Lu. Delay Guaranteed Energy-Efficient Computation Offloading for Industrial IoT in Fog Computing. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, Shanghai, China, May 2019. IEEE.

[12] Nhu-Ngoc Dao, Duc-Nghia Vu, Yunseong Lee, Sungrae Cho, Chihyun Cho, and Hyunbum Kim. Pattern-Identified Online Task Scheduling in Multitier Edge Computing for Industrial IoT Services. *Mobile Information Systems*, 2018:e2101206, April 2018. Publisher: Hindawi.

[13] Peter Drahoš, Erik Kucera, Oto Haffner, and Ivan Klimo. Trends in industrial communication and OPC UA. pages 1–5, January 2018.

[14] Juan Fang, Jiamei Shi, Shuaibing Lu, Mengyuan Zhang, and Zhiyuan Ye. An Efficient Computation Offloading Strategy with Mobile Edge Computing for IoT. *Micromachines*, 12(2), February 2021.

[15] Huber Flores, Xiang Su, Vassilis Kostakos, Aaron Yi Ding, Petteri Nurmi, Sasu Tarkoma, Pan Hui, and Yong Li. Large-scale offloading in the Internet of Things. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 479–484, Kona, HI, March 2017. IEEE.

[16] Xin Gao, Xi Huang, Simeng Bian, Ziyu Shao, and Yang Yang. PORA: Predictive Offloading and Resource Allocation in Dynamic Fog Computing Systems. *IEEE Internet of Things Journal*, 7(1):72–87, January 2020. arXiv: 2008.00204.

[17] Xiaoyu Hao, Ruohai Zhao, Tao Yang, Yulin Hu, Bo Hu, and Yuhe Qiu. *A Risk-Sensitive Task Offloading Strategy for Edge Computing in Industrial Internet of Things*. October 2020.

[18] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[19] Zicong Hong, Wuhui Chen, Huawei Huang, Song Guo, and Zibin Zheng. Multi-Hop Cooperative Computation Offloading for Industrial IoT–Edge–Cloud Computing Environments. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2759–2774, December 2019. Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[20] Md. Sajjad Hossain, Cosmas Ifeanyi Nwakanma, Jae Min Lee, and Dong-Seong Kim. Edge computational task offloading scheme using reinforcement learning for IIoT scenario. *ICT Express*, 6(4):291–299, December 2020.

[21] Ben Hoyle, Markus Michael Rau, Kerstin Paech, Christopher Bonnett, Stella Seitz, and Jochen Weller. Anomaly detection for machine learning redshifts applied to SDSS galaxies. *Monthly Notices of the Royal Astronomical Society*, 452(4):4183–4194, October 2015. arXiv: 1503.08214.

[22] Mohamed K. Hussein and Mohamed H. Mousa. Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization. *IEEE Access*, 8:37191–37201, 2020. Conference Name: IEEE Access.

[23] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, 7:133653–133667, 2019. Conference Name: IEEE Access.

[24] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[25] Sungwook Kim. New Application Task Offloading Algorithms for Edge, Fog, and Cloud Computing Paradigms. *Wireless Communications and Mobile Computing*, 2020:e8888074, October 2020. Publisher: Hindawi.

[26] Sachin Kumar, Prayag Tiwari, and Mikhail Zymbler. Internet of Things is a revolutionary approach for future technology enhancement: a review. *Journal of Big Data*, 6(1):111, December 2019.

[27] Phillip A. Laplante, Mohamad Kassab, Nancy L. Laplante, and Jeffrey M. Voas. Building Caring Healthcare Systems in the Internet of Things. *IEEE systems journal*, 12(3), 2018.

[28] Jiliang Li, Minghui Dai, and Zhou Su. Energy-Aware Task Offloading in the Internet of Things. *IEEE Wireless Communications*, 27(5):112–117, October 2020. Conference Name: IEEE Wireless Communications.

[29] Biswajeeban Mishra and Attila Kertesz. The Use of MQTT in M2M and IoT Systems: A Survey. *IEEE Access*, 8:201071–201086, 2020. Conference Name: IEEE Access.

[30] Peshawa Muhammad Ali and Rezhna Faraj. *Data Normalization and Standardization: A Technical Report*. January 2014.

[31] Eliseu Pereira, Joao Reis, and Gil Goncalves. DINASORE: A Dynamic Intelligent Reconfiguration Tool for Cyber-Physical Production Systems. page 9.

[32] Partha Pratim Ray. An Introduction to Dew Computing: Definition, Concept and Implications. *IEEE Access*, 6:723–737, 2018. Conference Name: IEEE Access.

[33] Maicon Saturno, Vinícius Pertel, and Fernando Deschamps. Proposal of an automation solutions architecture for Industry 4.0. July 2017.

[34] K R Srinath. Python – The Fastest Growing Programming Language. 04(12):4.

[35] Thomas Strasser, Martijn Rooker, Gerhard Ebenhofer, Alois Zoitl, Christoph Sunder, Antonio Valentini, and Allan Martel. Framework for Distributed Industrial Automation and Control (4DIAC). In *2008 6th IEEE International Conference on Industrial Informatics*, pages 283–288, July 2008. ISSN: 2378-363X.

[36] S Vidhya, Asir Antony Danasingh, and JEBAMALAR LEAVLINE EPIPHANY. Feature Extraction for Document Classification. April 2015.

# Chapter 8

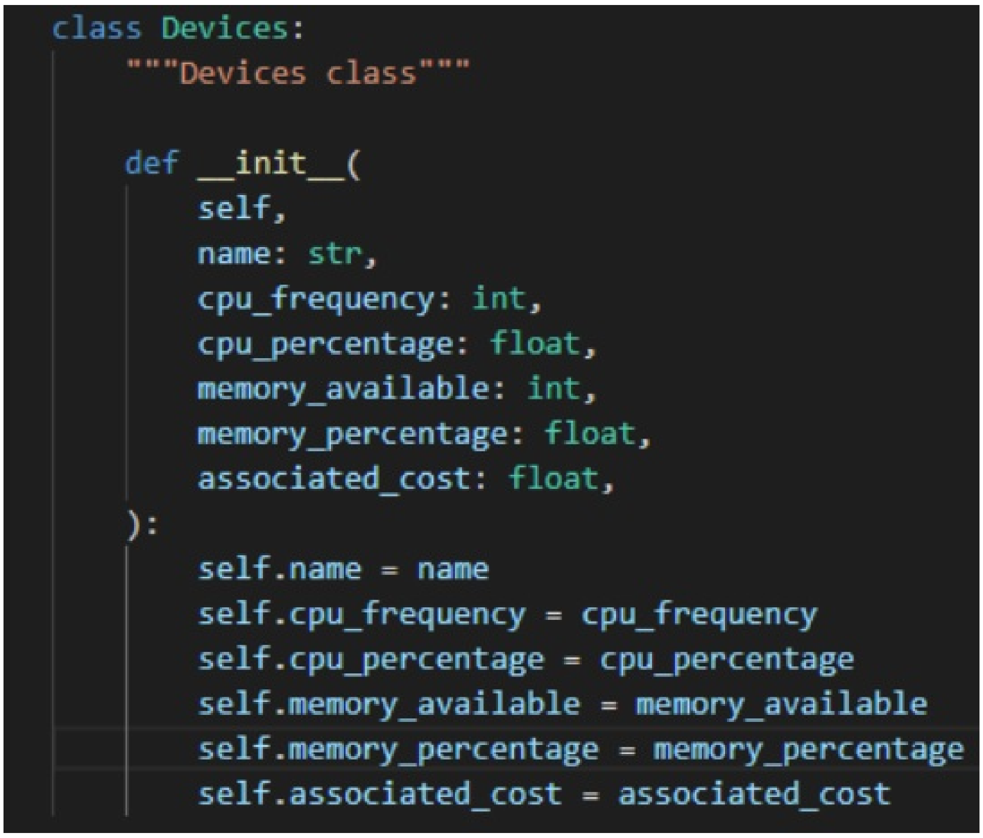
# Appendix

In this chapter figures related to the code will be displayed. The link to the spreadsheet of results is also made available.

## 8.1 Spreadsheet Link

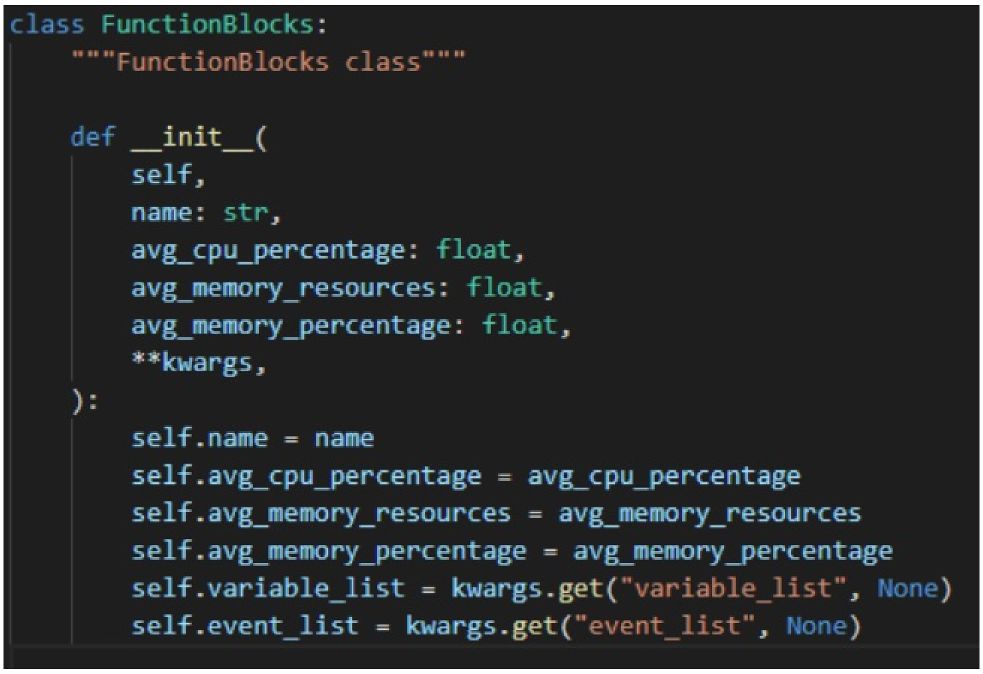
The spreadsheet whose data was filled in the tables presented in section 5.3 is available at: `https://uporto-my.sharepoint.com/:x:/g/personal/up201909575_up_pt/EQbLOjReRy1Css99Qj18PtUBvvOSGiZ_5Ov27uY79lZK6A?e=TP5J9k`

## 8.2 Code Related Figures



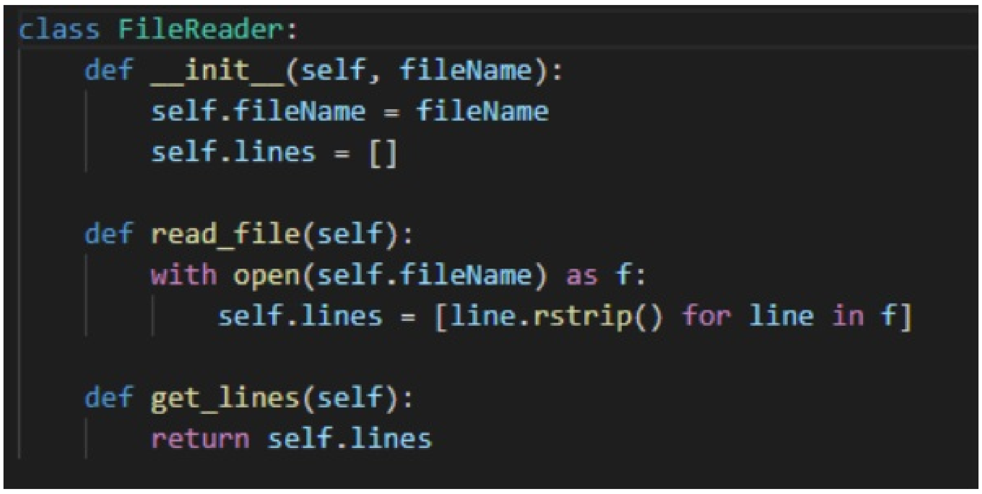
```python
class Devices:
    """Devices class"""

    def __init__(
        self,
        name: str,
        cpu_frequency: int,
        cpu_percentage: float,
        memory_available: int,
        memory_percentage: float,
        associated_cost: float,
    ):
        self.name = name
        self.cpu_frequency = cpu_frequency
        self.cpu_percentage = cpu_percentage
        self.memory_available = memory_available
        self.memory_percentage = memory_percentage
        self.associated_cost = associated_cost
```

Figure 8.1: Devices Class



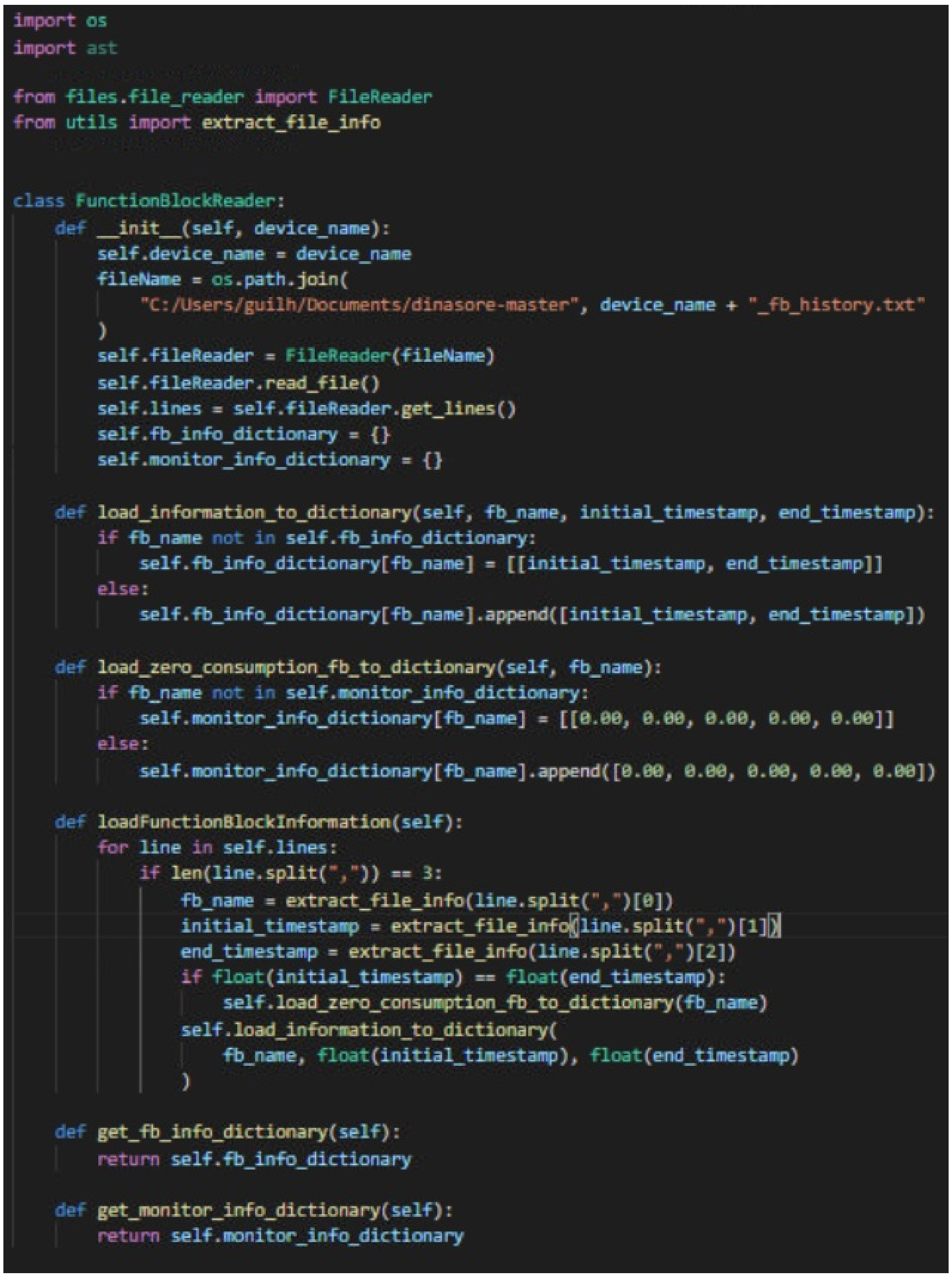
```python
class FunctionBlocks:
    """FunctionBlocks class"""

    def __init__(
        self,
        name: str,
        avg_cpu_percentage: float,
        avg_memory_resources: float,
        avg_memory_percentage: float,
        **kwargs,
    ):
        self.name = name
        self.avg_cpu_percentage = avg_cpu_percentage
        self.avg_memory_resources = avg_memory_resources
        self.avg_memory_percentage = avg_memory_percentage
        self.variable_list = kwargs.get("variable_list", None)
        self.event_list = kwargs.get("event_list", None)
```

Figure 8.2: Function Blocks Class



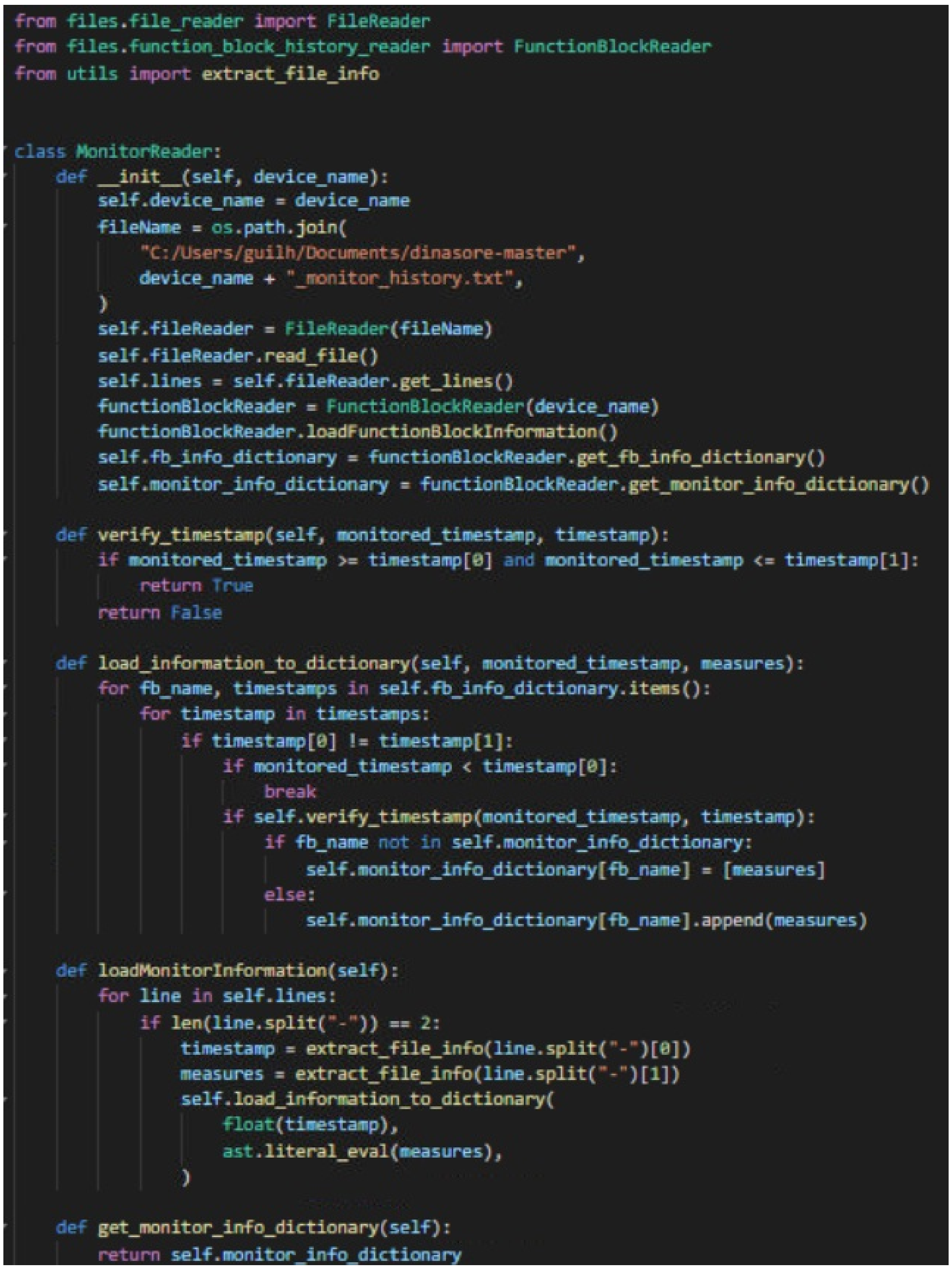
```python
class FileReader:
    def __init__(self, fileName):
        self.fileName = fileName
        self.lines = []

    def read_file(self):
        with open(self.fileName) as f:
            self.lines = [line.rstrip() for line in f]

    def get_lines(self):
        return self.lines
```

Figure 8.3: File Reader Class



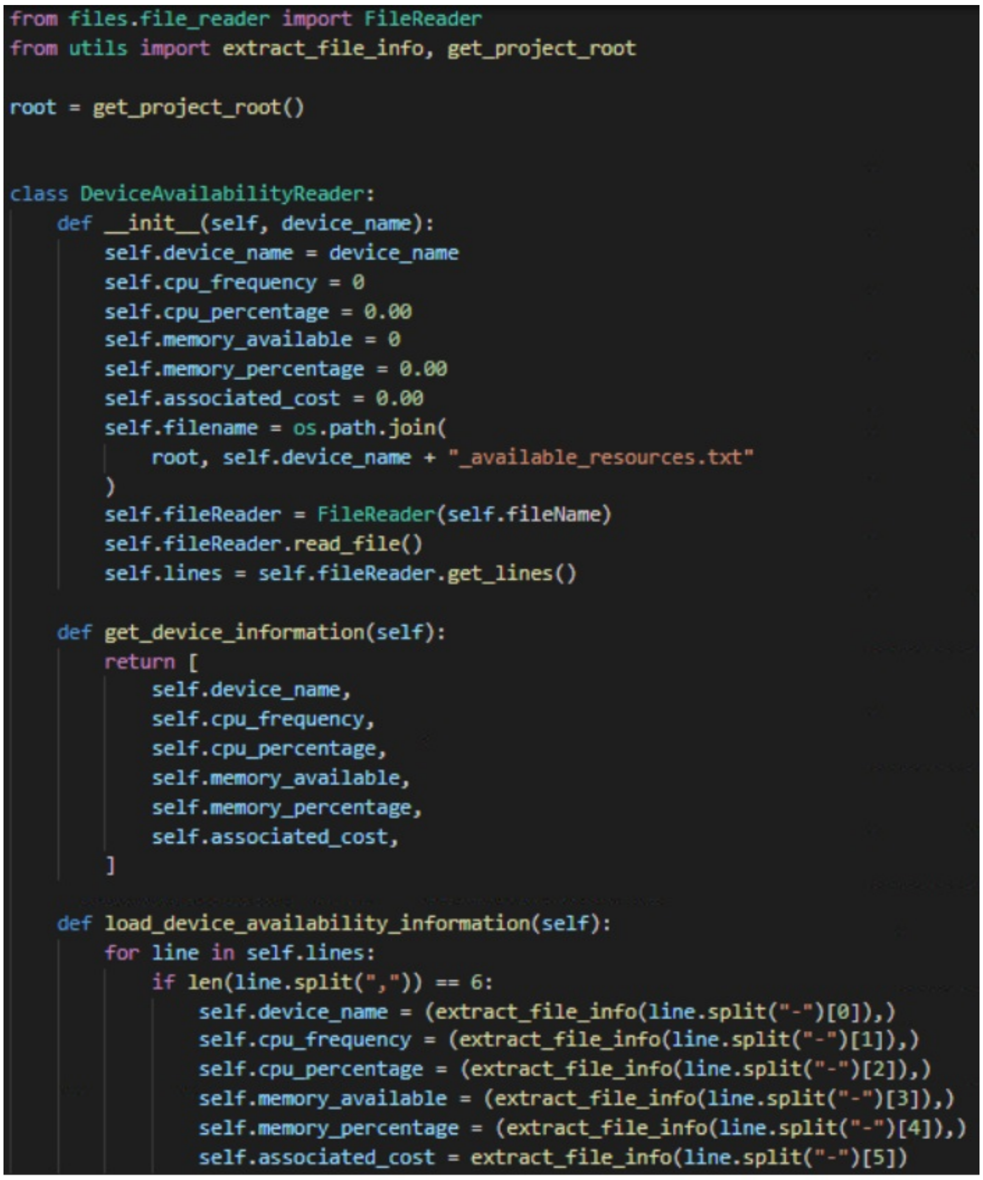
```python
import os
import ast

from files.file_reader import FileReader
from utils import extract_file_info


class FunctionBlockReader:
    def __init__(self, device_name):
        self.device_name = device_name
        fileName = os.path.join(
            "C:/Users/guilh/Documents/dinasore-master", device_name + "_fb_history.txt"
        )
        self.fileReader = FileReader(fileName)
        self.fileReader.read_file()
        self.lines = self.fileReader.get_lines()
        self.fb_info_dictionary = {}
        self.monitor_info_dictionary = {}

    def load_information_to_dictionary(self, fb_name, initial_timestamp, end_timestamp):
        if fb_name not in self.fb_info_dictionary:
            self.fb_info_dictionary[fb_name] = [[initial_timestamp, end_timestamp]]
        else:
            self.fb_info_dictionary[fb_name].append([initial_timestamp, end_timestamp])

    def load_zero_consumption_fb_to_dictionary(self, fb_name):
        if fb_name not in self.monitor_info_dictionary:
            self.monitor_info_dictionary[fb_name] = [[0.00, 0.00, 0.00, 0.00, 0.00]]
        else:
            self.monitor_info_dictionary[fb_name].append([0.00, 0.00, 0.00, 0.00, 0.00])

    def loadFunctionBlockInformation(self):
        for line in self.lines:
            if len(line.split(",")) == 3:
                fb_name = extract_file_info(line.split(",")[0])
                initial_timestamp = extract_file_info(line.split(",")[1])
                end_timestamp = extract_file_info(line.split(",")[2])
                if float(initial_timestamp) == float(end_timestamp):
                    self.load_zero_consumption_fb_to_dictionary(fb_name)
                self.load_information_to_dictionary(
                    fb_name, float(initial_timestamp), float(end_timestamp)
                )

    def get_fb_info_dictionary(self):
        return self.fb_info_dictionary

    def get_monitor_info_dictionary(self):
        return self.monitor_info_dictionary
```

Figure 8.4: Function Block Reader Class



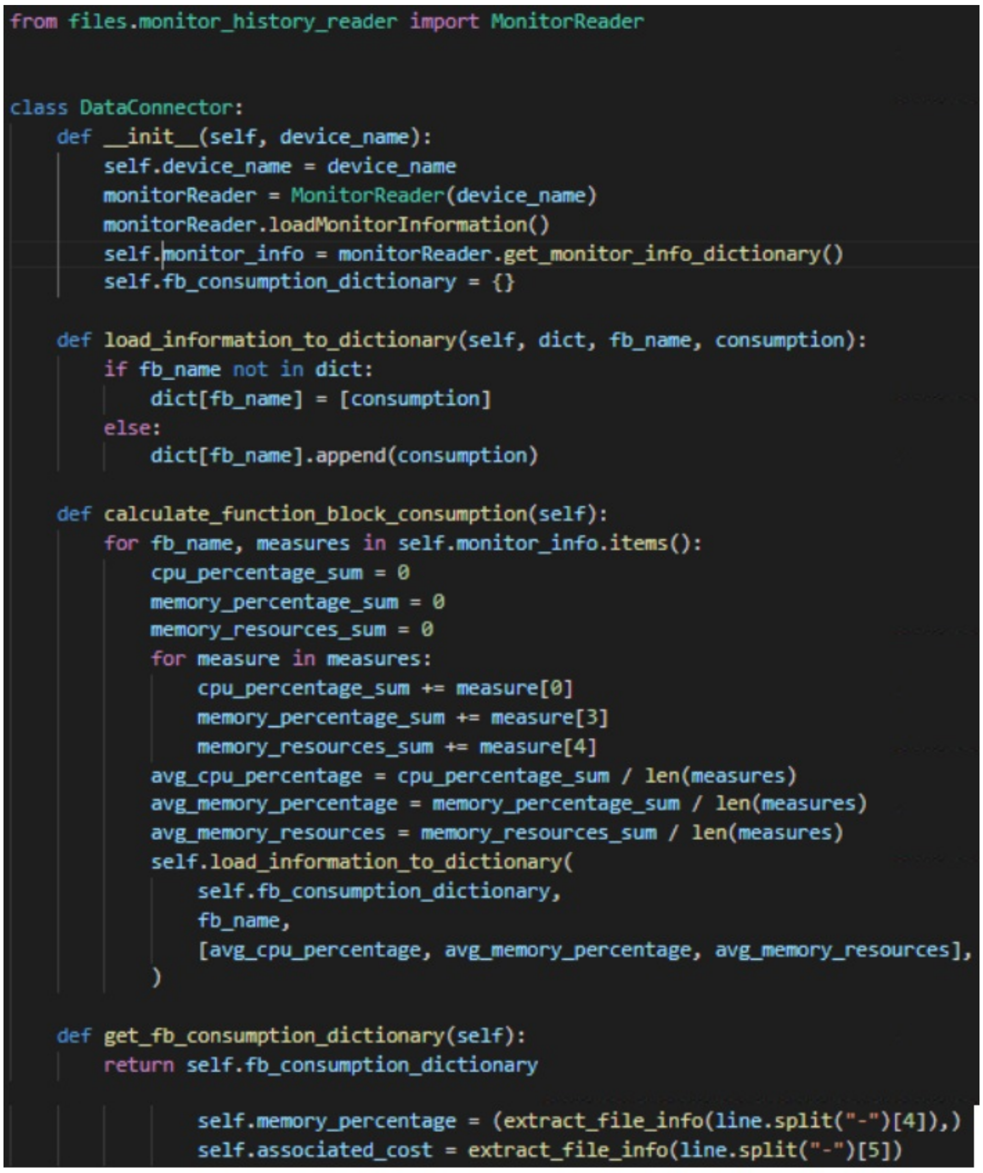
```python
from files.file_reader import FileReader
from files.function_block_history_reader import FunctionBlockReader
from utils import extract_file_info


class MonitorReader:
    def __init__(self, device_name):
        self.device_name = device_name
        fileName = os.path.join(
            "C:/Users/guilh/Documents/dinasore-master",
            device_name + "_monitor_history.txt",
        )
        self.fileReader = FileReader(fileName)
        self.fileReader.read_file()
        self.lines = self.fileReader.get_lines()
        functionBlockReader = FunctionBlockReader(device_name)
        functionBlockReader.loadFunctionBlockInformation()
        self.fb_info_dictionary = functionBlockReader.get_fb_info_dictionary()
        self.monitor_info_dictionary = functionBlockReader.get_monitor_info_dictionary()

    def verify_timestamp(self, monitored_timestamp, timestamp):
        if monitored_timestamp >= timestamp[0] and monitored_timestamp <= timestamp[1]:
            return True
        return False

    def load_information_to_dictionary(self, monitored_timestamp, measures):
        for fb_name, timestamps in self.fb_info_dictionary.items():
            for timestamp in timestamps:
                if timestamp[0] != timestamp[1]:
                    if monitored_timestamp < timestamp[0]:
                        break
                    if self.verify_timestamp(monitored_timestamp, timestamp):
                        if fb_name not in self.monitor_info_dictionary:
                            self.monitor_info_dictionary[fb_name] = [measures]
                        else:
                            self.monitor_info_dictionary[fb_name].append(measures)

    def loadMonitorInformation(self):
        for line in self.lines:
            if len(line.split("-")) == 2:
                timestamp = extract_file_info(line.split("-")[0])
                measures = extract_file_info(line.split("-")[1])
                self.load_information_to_dictionary(
                    float(timestamp),
                    ast.literal_eval(measures),
                )

    def get_monitor_info_dictionary(self):
        return self.monitor_info_dictionary
```

Figure 8.5: Monitor Reader Class



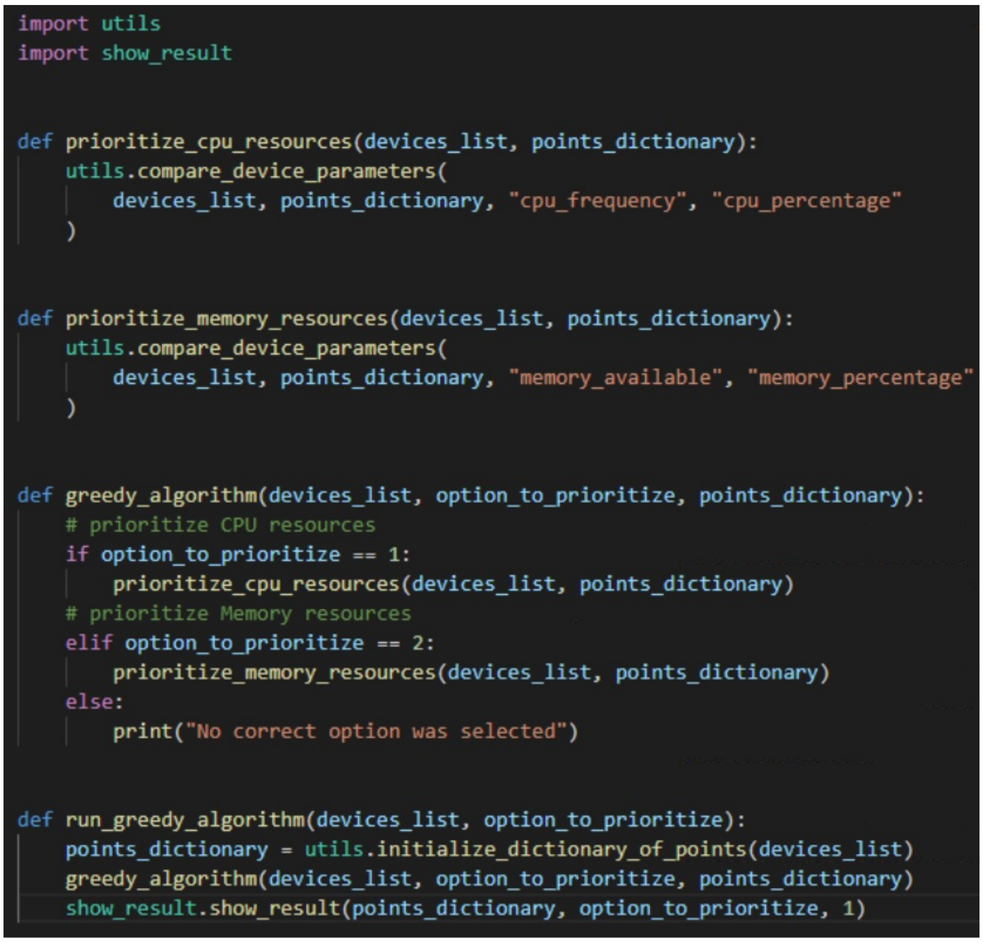
```python
from files.file_reader import FileReader
from utils import extract_file_info, get_project_root

root = get_project_root()


class DeviceAvailabilityReader:
    def __init__(self, device_name):
        self.device_name = device_name
        self.cpu_frequency = 0
        self.cpu_percentage = 0.00
        self.memory_available = 0
        self.memory_percentage = 0.00
        self.associated_cost = 0.00
        self.filename = os.path.join(
            root, self.device_name + "_available_resources.txt"
        )
        self.fileReader = FileReader(self.fileName)
        self.fileReader.read_file()
        self.lines = self.fileReader.get_lines()

    def get_device_information(self):
        return [
            self.device_name,
            self.cpu_frequency,
            self.cpu_percentage,
            self.memory_available,
            self.memory_percentage,
            self.associated_cost,
        ]

    def load_device_availability_information(self):
        for line in self.lines:
            if len(line.split(",")) == 6:
                self.device_name = (extract_file_info(line.split("-")[0]),)
                self.cpu_frequency = (extract_file_info(line.split("-")[1]),)
                self.cpu_percentage = (extract_file_info(line.split("-")[2]),)
                self.memory_available = (extract_file_info(line.split("-")[3]),)
                self.memory_percentage = (extract_file_info(line.split("-")[4]),)
                self.associated_cost = extract_file_info(line.split("-")[5])
```

Figure 8.6: Device Availability Reader Class



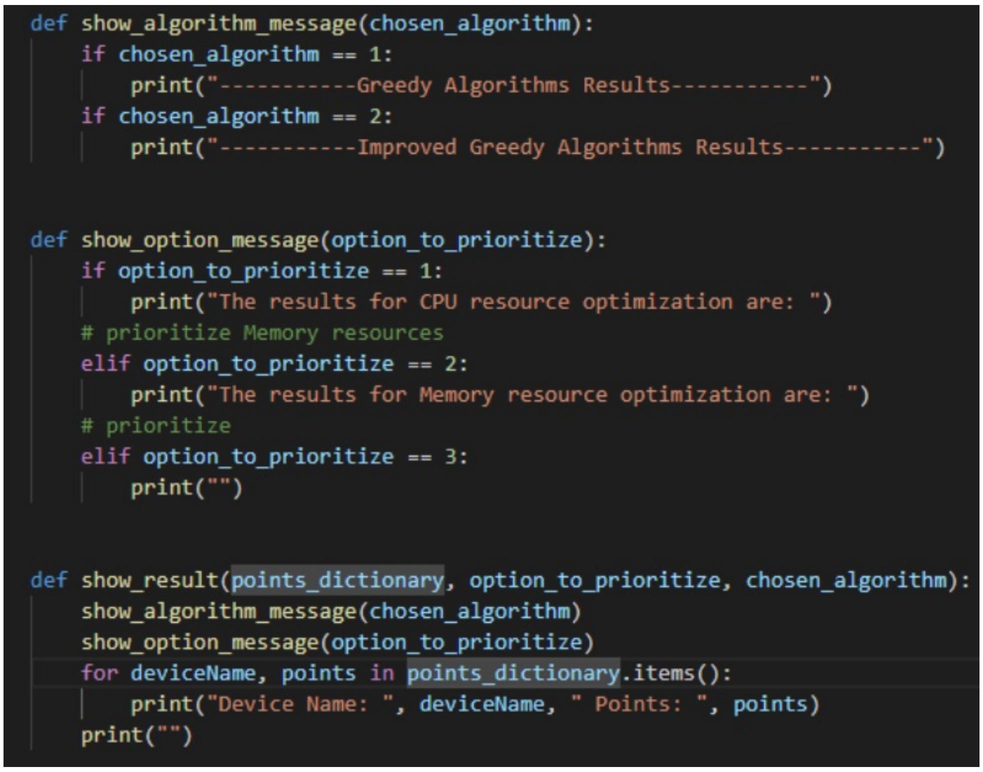
```
from files.monitor_history_reader import MonitorReader


class DataConnector:
    def __init__(self, device_name):
        self.device_name = device_name
        monitorReader = MonitorReader(device_name)
        monitorReader.loadMonitorInformation()
        self.monitor_info = monitorReader.get_monitor_info_dictionary()
        self.fb_consumption_dictionary = {}

    def load_information_to_dictionary(self, dict, fb_name, consumption):
        if fb_name not in dict:
            dict[fb_name] = [consumption]
        else:
            dict[fb_name].append(consumption)

    def calculate_function_block_consumption(self):
        for fb_name, measures in self.monitor_info.items():
            cpu_percentage_sum = 0
            memory_percentage_sum = 0
            memory_resources_sum = 0
            for measure in measures:
                cpu_percentage_sum += measure[0]
                memory_percentage_sum += measure[3]
                memory_resources_sum += measure[4]
            avg_cpu_percentage = cpu_percentage_sum / len(measures)
            avg_memory_percentage = memory_percentage_sum / len(measures)
            avg_memory_resources = memory_resources_sum / len(measures)
            self.load_information_to_dictionary(
                self.fb_consumption_dictionary,
                fb_name,
                [avg_cpu_percentage, avg_memory_percentage, avg_memory_resources],
            )

    def get_fb_consumption_dictionary(self):
        return self.fb_consumption_dictionary

                self.memory_percentage = (extract_file_info(line.split("-")[4]),)
                self.associated_cost = extract_file_info(line.split("-")[5])
```

Figure 8.7: Data Connector Class



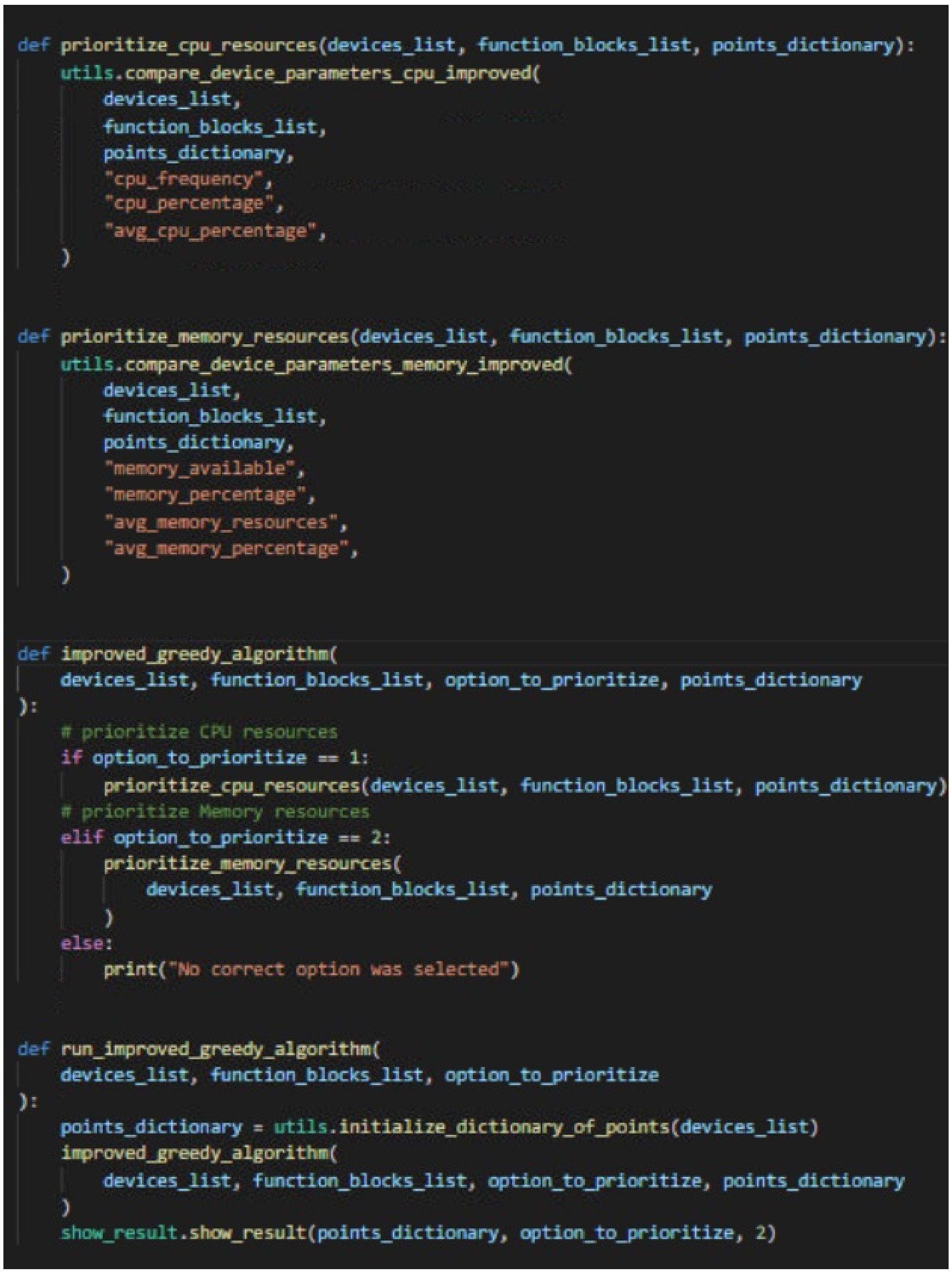
```python
import utils
import show_result


def prioritize_cpu_resources(devices_list, points_dictionary):
    utils.compare_device_parameters(
        devices_list, points_dictionary, "cpu_frequency", "cpu_percentage"
    )


def prioritize_memory_resources(devices_list, points_dictionary):
    utils.compare_device_parameters(
        devices_list, points_dictionary, "memory_available", "memory_percentage"
    )


def greedy_algorithm(devices_list, option_to_prioritize, points_dictionary):
    # prioritize CPU resources
    if option_to_prioritize == 1:
        prioritize_cpu_resources(devices_list, points_dictionary)
    # prioritize Memory resources
    elif option_to_prioritize == 2:
        prioritize_memory_resources(devices_list, points_dictionary)
    else:
        print("No correct option was selected")


def run_greedy_algorithm(devices_list, option_to_prioritize):
    points_dictionary = utils.initialize_dictionary_of_points(devices_list)
    greedy_algorithm(devices_list, option_to_prioritize, points_dictionary)
    show_result.show_result(points_dictionary, option_to_prioritize, 1)
```

Figure 8.8: Greedy Algorithm Class



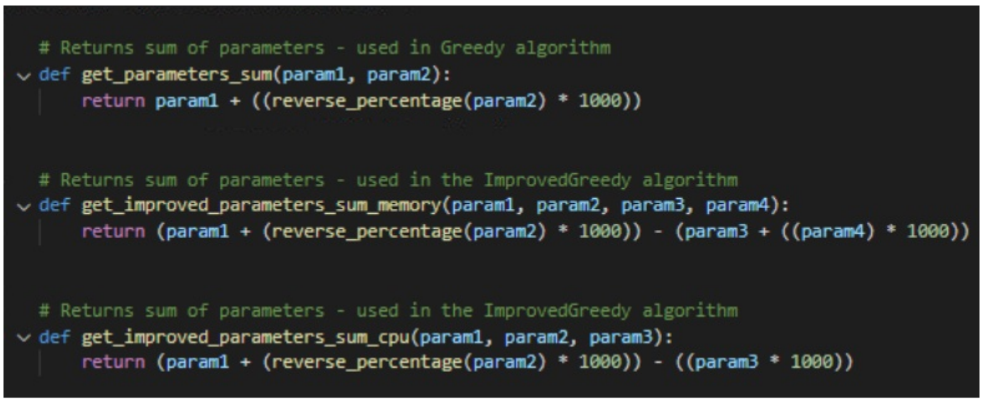
```python
def show_algorithm_message(chosen_algorithm):
    if chosen_algorithm == 1:
        print("-----------Greedy Algorithms Results-----------")
    if chosen_algorithm == 2:
        print("-----------Improved Greedy Algorithms Results-----------")


def show_option_message(option_to_prioritize):
    if option_to_prioritize == 1:
        print("The results for CPU resource optimization are: ")
    # prioritize Memory resources
    elif option_to_prioritize == 2:
        print("The results for Memory resource optimization are: ")
    # prioritize
    elif option_to_prioritize == 3:
        print("")


def show_result(points_dictionary, option_to_prioritize, chosen_algorithm):
    show_algorithm_message(chosen_algorithm)
    show_option_message(option_to_prioritize)
    for deviceName, points in points_dictionary.items():
        print("Device Name: ", deviceName, " Points: ", points)
    print("")
```

Figure 8.9: Show Result Class



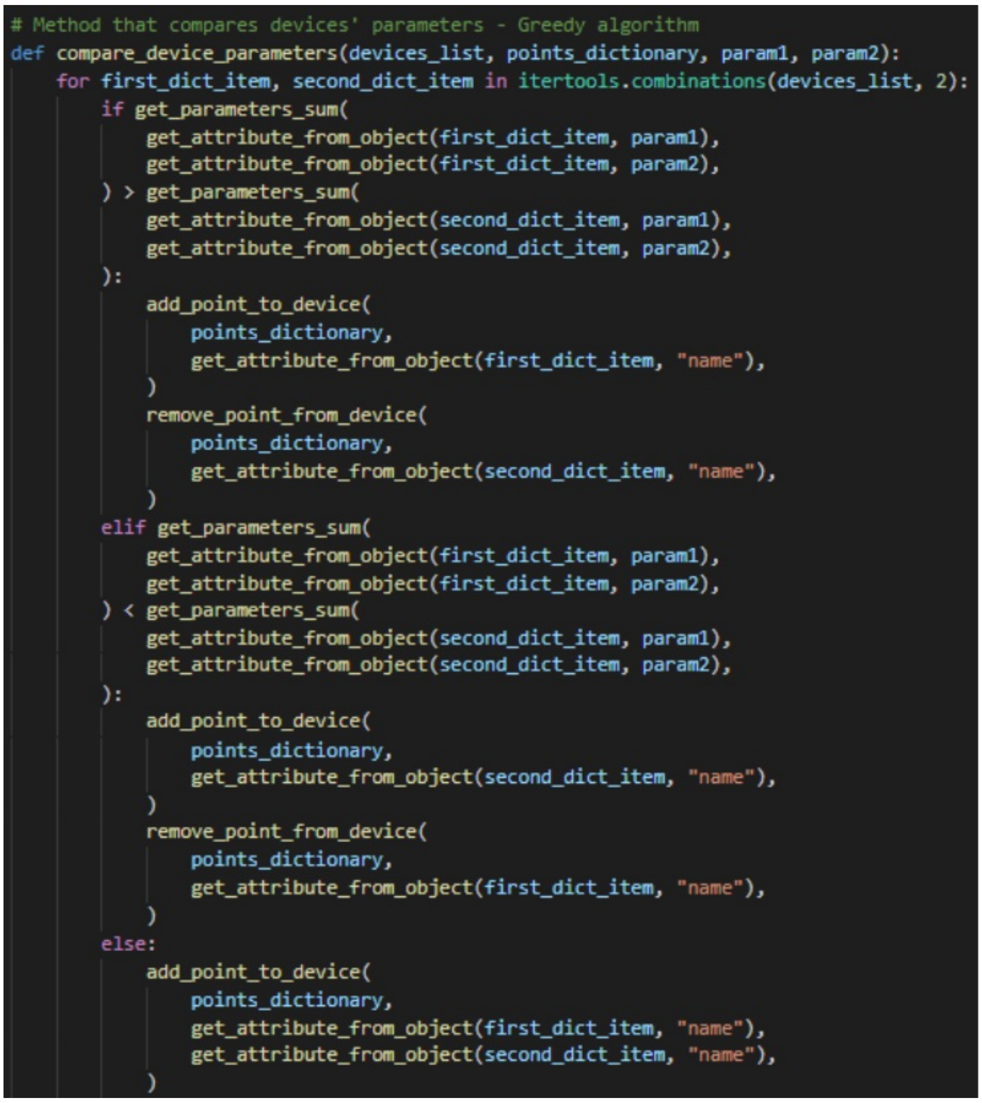
```python
def prioritize_cpu_resources(devices_list, function_blocks_list, points_dictionary):
    utils.compare_device_parameters_cpu_improved(
        devices_list,
        function_blocks_list,
        points_dictionary,
        "cpu_frequency",
        "cpu_percentage",
        "avg_cpu_percentage",
    )


def prioritize_memory_resources(devices_list, function_blocks_list, points_dictionary):
    utils.compare_device_parameters_memory_improved(
        devices_list,
        function_blocks_list,
        points_dictionary,
        "memory_available",
        "memory_percentage",
        "avg_memory_resources",
        "avg_memory_percentage",
    )


def improved_greedy_algorithm(
    devices_list, function_blocks_list, option_to_prioritize, points_dictionary
):
    # prioritize CPU resources
    if option_to_prioritize == 1:
        prioritize_cpu_resources(devices_list, function_blocks_list, points_dictionary)
    # prioritize Memory resources
    elif option_to_prioritize == 2:
        prioritize_memory_resources(
            devices_list, function_blocks_list, points_dictionary
        )
    else:
        print("No correct option was selected")


def run_improved_greedy_algorithm(
    devices_list, function_blocks_list, option_to_prioritize
):
    points_dictionary = utils.initialize_dictionary_of_points(devices_list)
    improved_greedy_algorithm(
        devices_list, function_blocks_list, option_to_prioritize, points_dictionary
    )
    show_result.show_result(points_dictionary, option_to_prioritize, 2)
```

Figure 8.10: Improved Greedy Algorithm Class



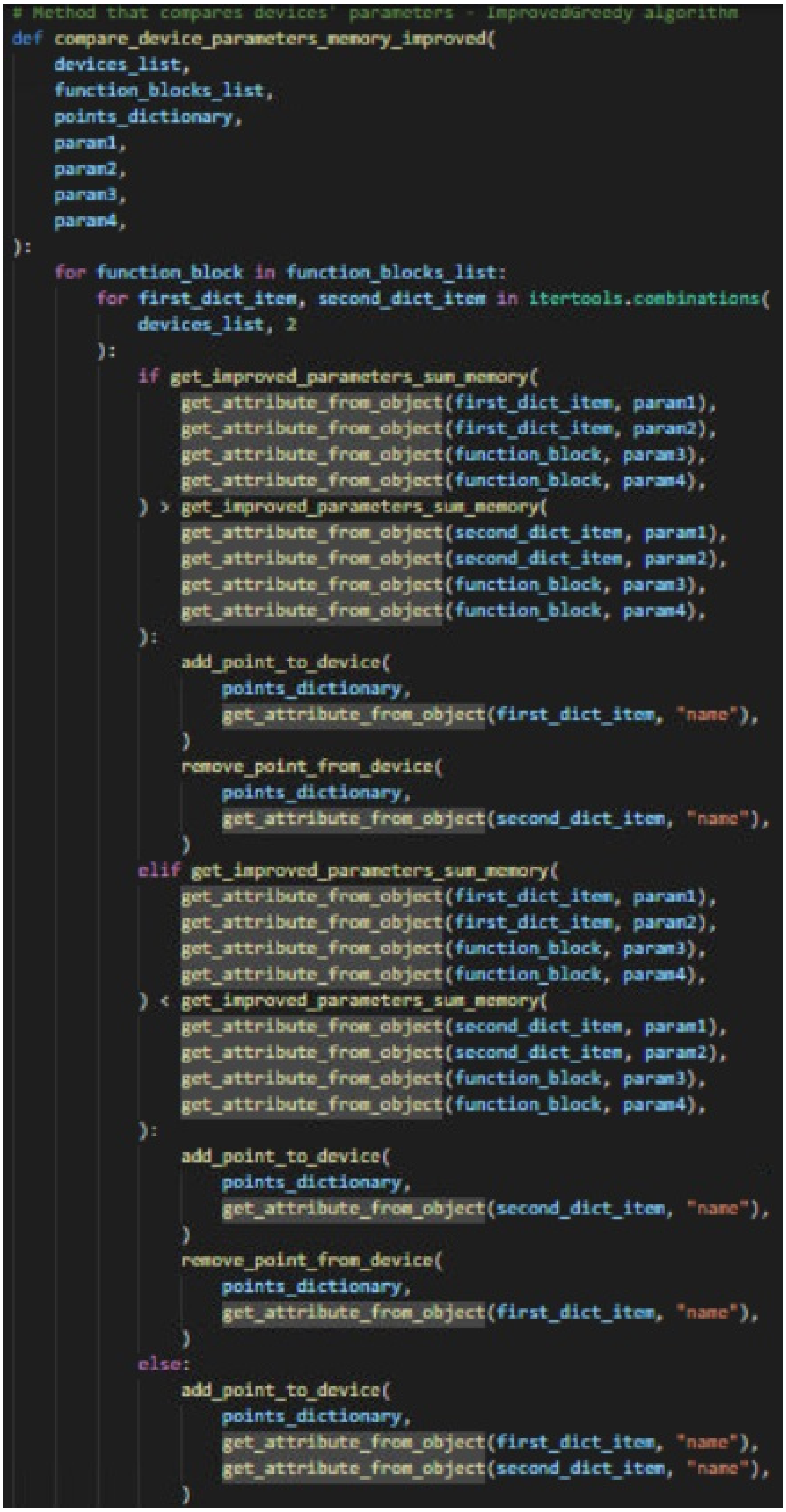
```python
# Returns sum of parameters - used in Greedy algorithm
def get_parameters_sum(param1, param2):
    return param1 + ((reverse_percentage(param2) * 1000))


# Returns sum of parameters - used in the ImprovedGreedy algorithm
def get_improved_parameters_sum_memory(param1, param2, param3, param4):
    return (param1 + (reverse_percentage(param2) * 1000)) - (param3 + ((param4) * 1000))


# Returns sum of parameters - used in the ImprovedGreedy algorithm
def get_improved_parameters_sum_cpu(param1, param2, param3):
    return (param1 + (reverse_percentage(param2) * 1000)) - ((param3 * 1000))
```

Figure 8.11: Methods utilized in the Greedy and Improved Greedy algorithms to calculate resource consumption



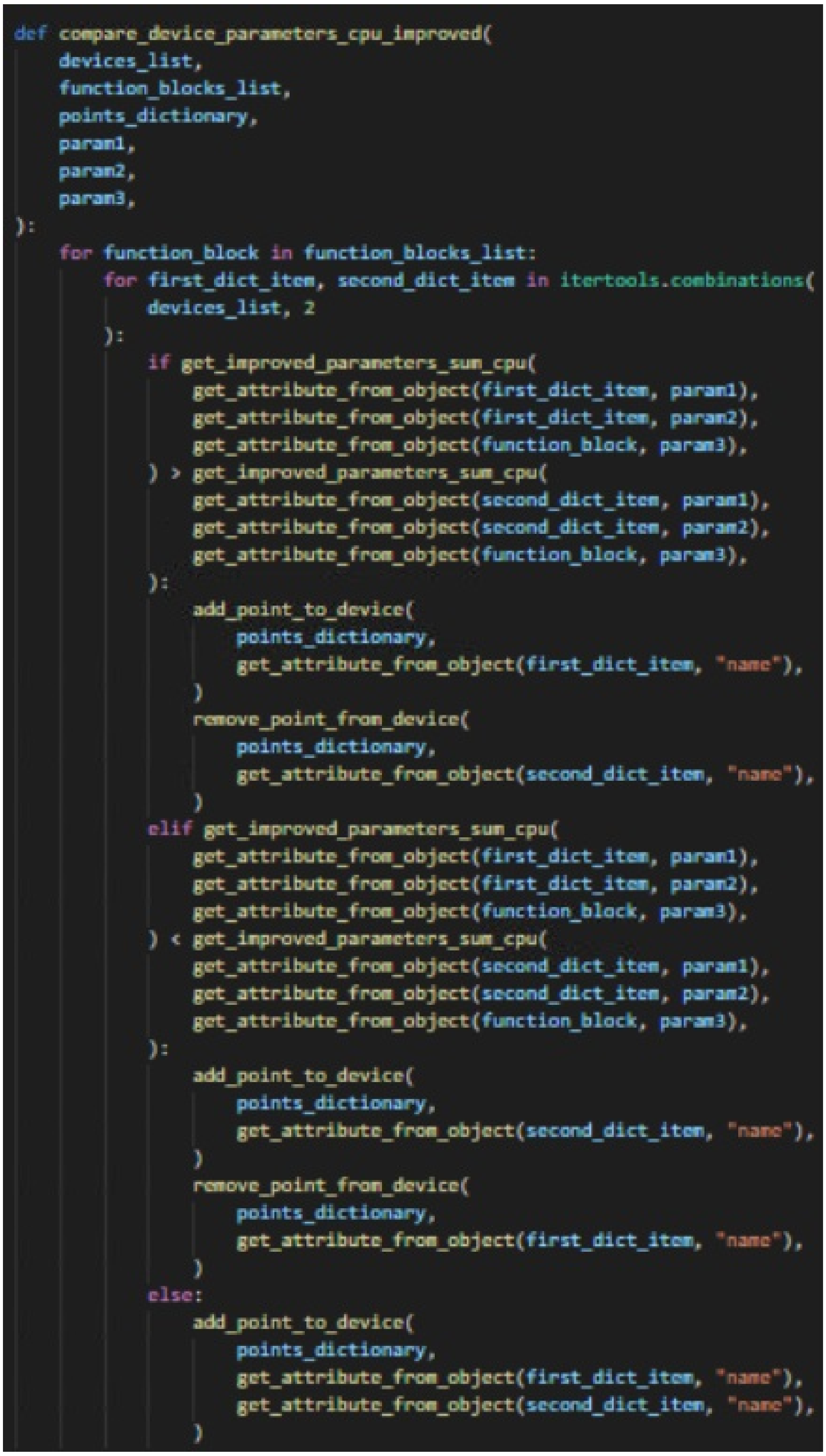
```python
# Method that compares devices' parameters - Greedy algorithm
def compare_device_parameters(devices_list, points_dictionary, param1, param2):
    for first_dict_item, second_dict_item in itertools.combinations(devices_list, 2):
        if get_parameters_sum(
            get_attribute_from_object(first_dict_item, param1),
            get_attribute_from_object(first_dict_item, param2),
        ) > get_parameters_sum(
            get_attribute_from_object(second_dict_item, param1),
            get_attribute_from_object(second_dict_item, param2),
        ):
            add_point_to_device(
                points_dictionary,
                get_attribute_from_object(first_dict_item, "name"),
            )
            remove_point_from_device(
                points_dictionary,
                get_attribute_from_object(second_dict_item, "name"),
            )
        elif get_parameters_sum(
            get_attribute_from_object(first_dict_item, param1),
            get_attribute_from_object(first_dict_item, param2),
        ) < get_parameters_sum(
            get_attribute_from_object(second_dict_item, param1),
            get_attribute_from_object(second_dict_item, param2),
        ):
            add_point_to_device(
                points_dictionary,
                get_attribute_from_object(second_dict_item, "name"),
            )
            remove_point_from_device(
                points_dictionary,
                get_attribute_from_object(first_dict_item, "name"),
            )
        else:
            add_point_to_device(
                points_dictionary,
                get_attribute_from_object(first_dict_item, "name"),
                get_attribute_from_object(second_dict_item, "name"),
            )
```

Figure 8.12: Method utilized in the Greedy algorithm to obtain device comparison



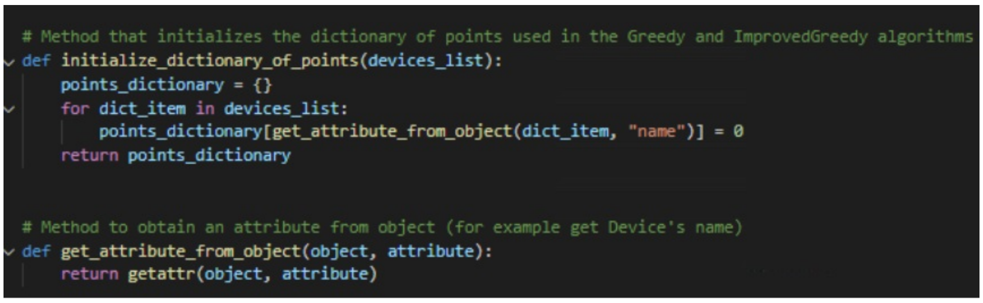
```python
# Method that compares devices' parameters - ImprovedGreedy algorithm
def compare_device_parameters_memory_improved(
    devices_list,
    function_blocks_list,
    points_dictionary,
    param1,
    param2,
    param3,
    param4,
):
    for function_block in function_blocks_list:
        for first_dict_item, second_dict_item in itertools.combinations(
            devices_list, 2
        ):
            if get_improved_parameters_sum_memory(
                get_attribute_from_object(first_dict_item, param1),
                get_attribute_from_object(first_dict_item, param2),
                get_attribute_from_object(function_block, param3),
                get_attribute_from_object(function_block, param4),
            ) > get_improved_parameters_sum_memory(
                get_attribute_from_object(second_dict_item, param1),
                get_attribute_from_object(second_dict_item, param2),
                get_attribute_from_object(function_block, param3),
                get_attribute_from_object(function_block, param4),
            ):
                add_point_to_device(
                    points_dictionary,
                    get_attribute_from_object(first_dict_item, "name"),
                )
                remove_point_from_device(
                    points_dictionary,
                    get_attribute_from_object(second_dict_item, "name"),
                )
            elif get_improved_parameters_sum_memory(
                get_attribute_from_object(first_dict_item, param1),
                get_attribute_from_object(first_dict_item, param2),
                get_attribute_from_object(function_block, param3),
                get_attribute_from_object(function_block, param4),
            ) < get_improved_parameters_sum_memory(
                get_attribute_from_object(second_dict_item, param1),
                get_attribute_from_object(second_dict_item, param2),
                get_attribute_from_object(function_block, param3),
                get_attribute_from_object(function_block, param4),
            ):
                add_point_to_device(
                    points_dictionary,
                    get_attribute_from_object(second_dict_item, "name"),
                )
                remove_point_from_device(
                    points_dictionary,
                    get_attribute_from_object(first_dict_item, "name"),
                )
            else:
                add_point_to_device(
                    points_dictionary,
                    get_attribute_from_object(first_dict_item, "name"),
                    get_attribute_from_object(second_dict_item, "name"),
                )
```

Figure 8.13: Method utilized in the Improved Greedy algorithm to obtain device comparison for memory resources



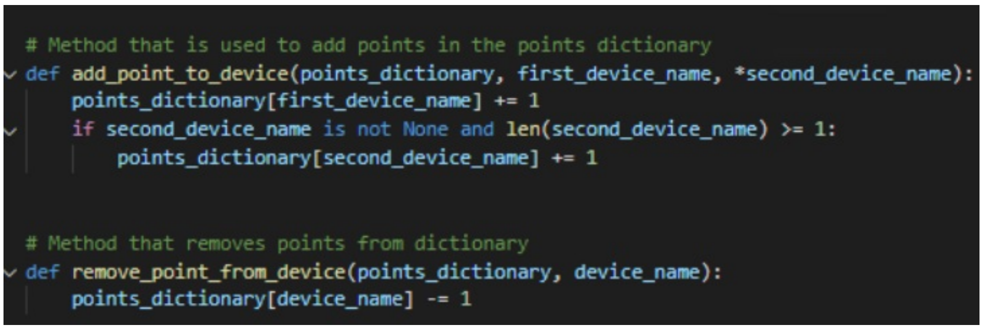
```python
def compare_device_parameters_cpu_improved(
    devices_list,
    function_blocks_list,
    points_dictionary,
    param1,
    param2,
    param3,
):
    for function_block in function_blocks_list:
        for first_dict_item, second_dict_item in itertools.combinations(
            devices_list, 2
        ):
            if get_improved_parameters_sum_cpu(
                get_attribute_from_object(first_dict_item, param1),
                get_attribute_from_object(first_dict_item, param2),
                get_attribute_from_object(function_block, param3),
            ) > get_improved_parameters_sum_cpu(
                get_attribute_from_object(second_dict_item, param1),
                get_attribute_from_object(second_dict_item, param2),
                get_attribute_from_object(function_block, param3),
            ):
                add_point_to_device(
                    points_dictionary,
                    get_attribute_from_object(first_dict_item, "name"),
                )
                remove_point_from_device(
                    points_dictionary,
                    get_attribute_from_object(second_dict_item, "name"),
                )
            elif get_improved_parameters_sum_cpu(
                get_attribute_from_object(first_dict_item, param1),
                get_attribute_from_object(first_dict_item, param2),
                get_attribute_from_object(function_block, param3),
            ) < get_improved_parameters_sum_cpu(
                get_attribute_from_object(second_dict_item, param1),
                get_attribute_from_object(second_dict_item, param2),
                get_attribute_from_object(function_block, param3),
            ):
                add_point_to_device(
                    points_dictionary,
                    get_attribute_from_object(second_dict_item, "name"),
                )
                remove_point_from_device(
                    points_dictionary,
                    get_attribute_from_object(first_dict_item, "name"),
                )
            else:
                add_point_to_device(
                    points_dictionary,
                    get_attribute_from_object(first_dict_item, "name"),
                    get_attribute_from_object(second_dict_item, "name"),
                )
```

Figure 8.14: Method utilized in the Improved Greedy algorithm to obtain device comparison for CPU resources



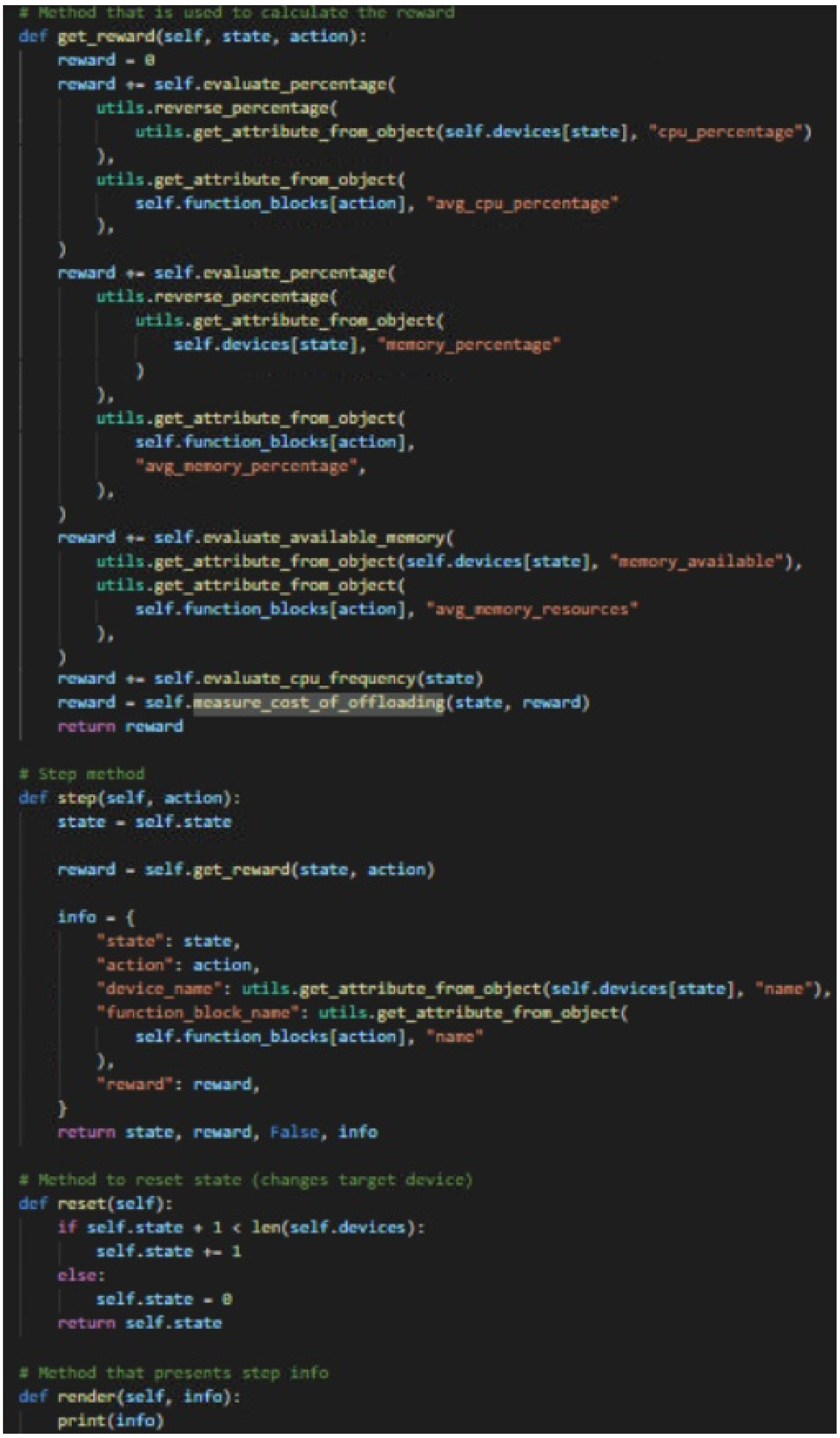
```python
# Method that initializes the dictionary of points used in the Greedy and ImprovedGreedy algorithms
def initialize_dictionary_of_points(devices_list):
    points_dictionary = {}
    for dict_item in devices_list:
        points_dictionary[get_attribute_from_object(dict_item, "name")] = 0
    return points_dictionary


# Method to obtain an attribute from object (for example get Device's name)
def get_attribute_from_object(object, attribute):
    return getattr(object, attribute)
```

Figure 8.15: Methods utilized in the Greedy and Improved Greedy algorithms to aid in the process of initializing dictionaries and obtaining an objects parameter



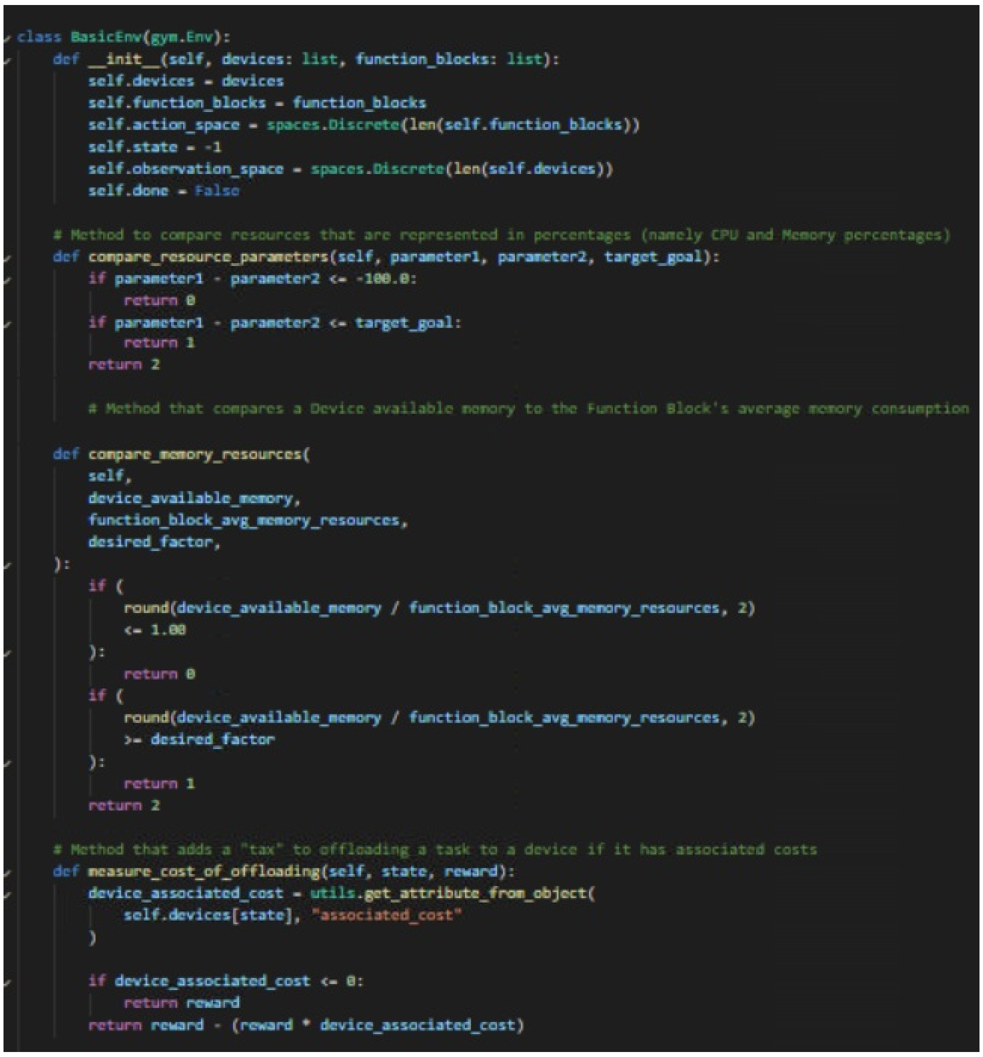
```python
# Method that is used to add points in the points dictionary
def add_point_to_device(points_dictionary, first_device_name, *second_device_name):
    points_dictionary[first_device_name] += 1
    if second_device_name is not None and len(second_device_name) >= 1:
        points_dictionary[second_device_name] += 1


# Method that removes points from dictionary
def remove_point_from_device(points_dictionary, device_name):
    points_dictionary[device_name] -= 1
```

Figure 8.16: Methods utilized in the Greedy and Improved Greedy algorithms to aid in the process of manipulating dictionary data



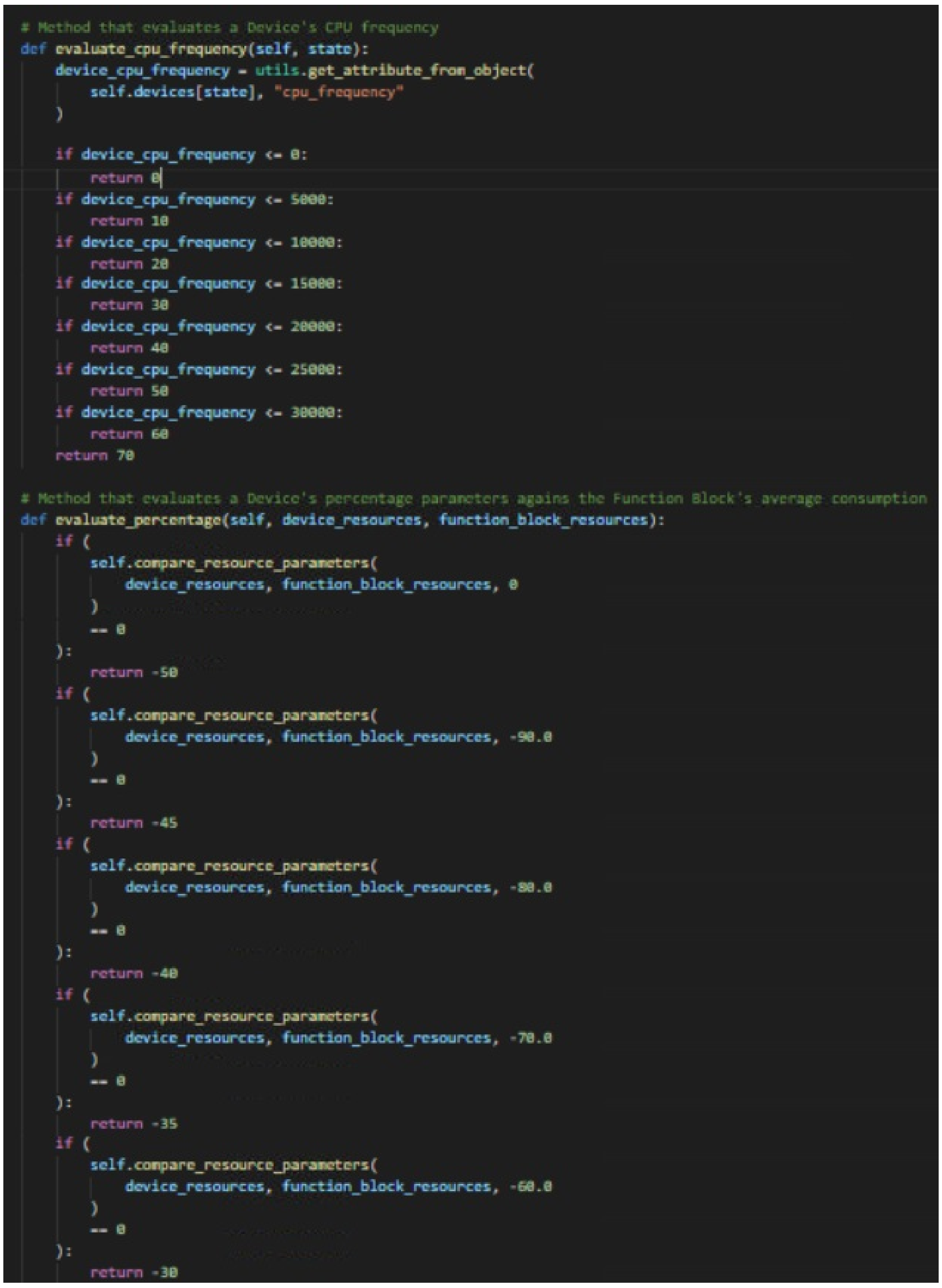
```
# Method that is used to calculate the reward
def get_reward(self, state, action):
    reward = 0
    reward += self.evaluate_percentage(
        utils.reverse_percentage(
            utils.get_attribute_from_object(self.devices[state], "cpu_percentage")
        ),
        utils.get_attribute_from_object(
            self.function_blocks[action], "avg_cpu_percentage"
        ),
    )
    reward += self.evaluate_percentage(
        utils.reverse_percentage(
            utils.get_attribute_from_object(
                self.devices[state], "memory_percentage"
            )
        ),
        utils.get_attribute_from_object(
            self.function_blocks[action],
            "avg_memory_percentage",
        ),
    )
    reward += self.evaluate_available_memory(
        utils.get_attribute_from_object(self.devices[state], "memory_available"),
        utils.get_attribute_from_object(
            self.function_blocks[action], "avg_memory_resources"
        ),
    )
    reward += self.evaluate_cpu_frequency(state)
    reward = self.measure_cost_of_offloading(state, reward)
    return reward

# Step method
def step(self, action):
    state = self.state

    reward = self.get_reward(state, action)

    info = {
        "state": state,
        "action": action,
        "device_name": utils.get_attribute_from_object(self.devices[state], "name"),
        "function_block_name": utils.get_attribute_from_object(
            self.function_blocks[action], "name"
        ),
        "reward": reward,
    }
    return state, reward, False, info

# Method to reset state (changes target device)
def reset(self):
    if self.state + 1 < len(self.devices):
        self.state += 1
    else:
        self.state = 0
    return self.state

# Method that presents step info
def render(self, info):
    print(info)
```

Figure 8.17: Methods utilized in the Q-Learning algorithm's environment to obtain reward, to calculate step, to render render and reset the algorithm's state



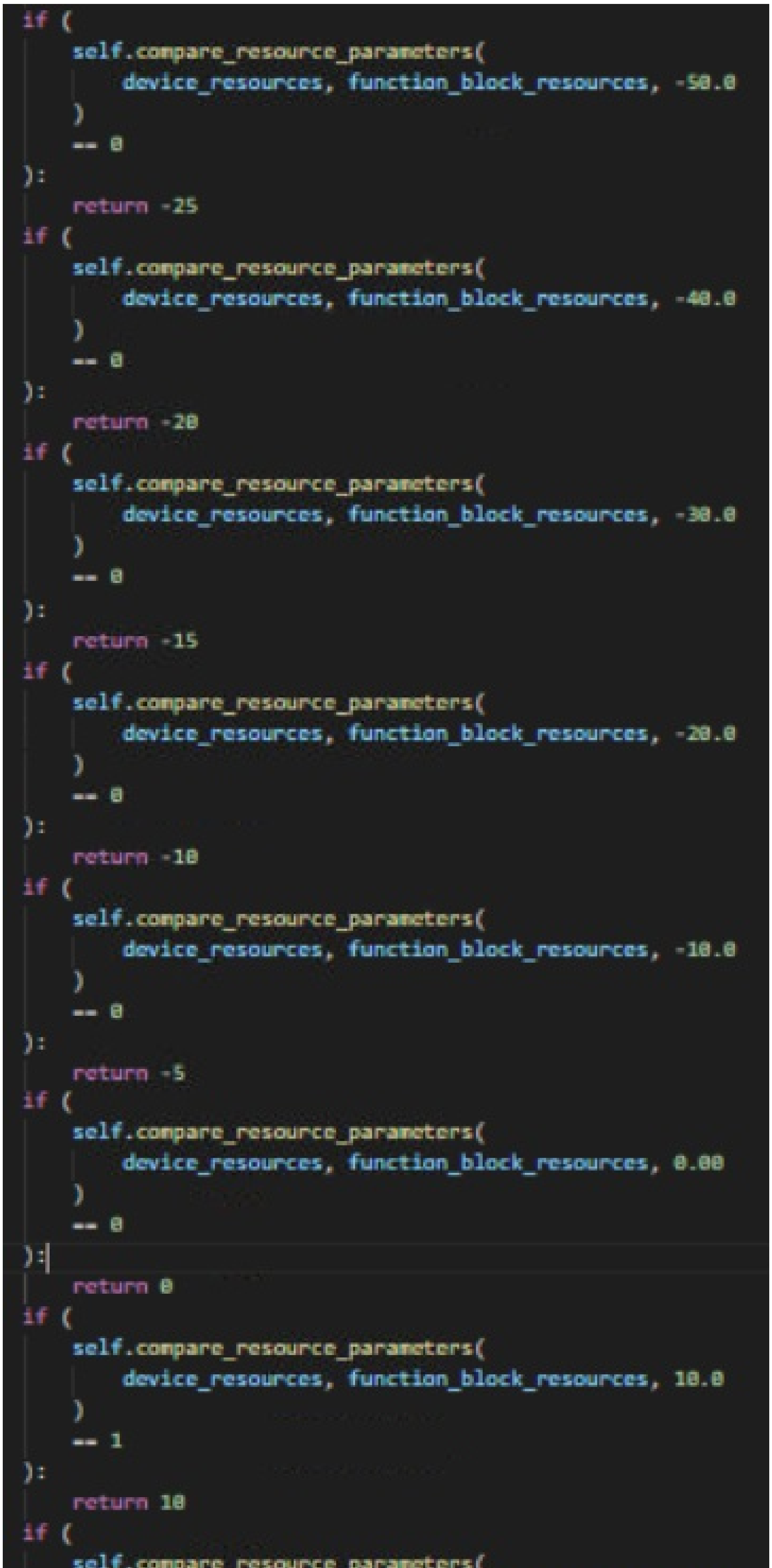
```python
class BasicEnv(gym.Env):
    def __init__(self, devices: list, function_blocks: list):
        self.devices = devices
        self.function_blocks = function_blocks
        self.action_space = spaces.Discrete(len(self.function_blocks))
        self.state = -1
        self.observation_space = spaces.Discrete(len(self.devices))
        self.done = False

    # Method to compare resources that are represented in percentages (namely CPU and Memory percentages)
    def compare_resource_parameters(self, parameter1, parameter2, target_goal):
        if parameter1 - parameter2 <= -100.0:
            return 0
        if parameter1 - parameter2 <= target_goal:
            return 1
        return 2

        # Method that compares a Device available memory to the Function Block's average memory consumption

    def compare_memory_resources(
        self,
        device_available_memory,
        function_block_avg_memory_resources,
        desired_factor,
    ):
        if (
            round(device_available_memory / function_block_avg_memory_resources, 2)
            <= 1.00
        ):
            return 0
        if (
            round(device_available_memory / function_block_avg_memory_resources, 2)
            >= desired_factor
        ):
            return 1
        return 2

    # Method that adds a "tax" to offloading a task to a device if it has associated costs
    def measure_cost_of_offloading(self, state, reward):
        device_associated_cost = utils.get_attribute_from_object(
            self.devices[state], "associated_cost"
        )

        if device_associated_cost <= 0:
            return reward
        return reward - (reward * device_associated_cost)
```

Figure 8.18: Environment class initialization and reward associated methods

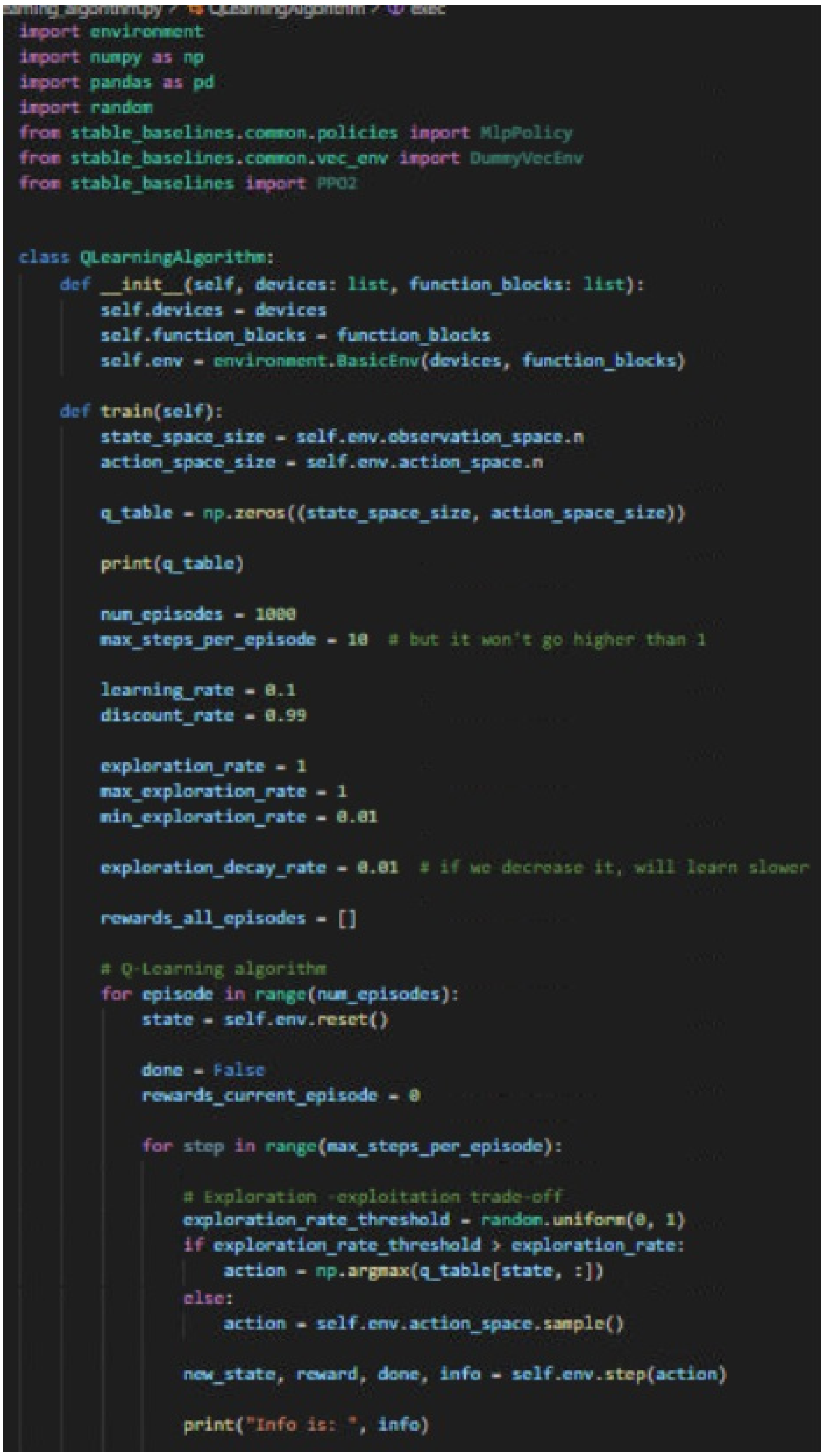


```python
# Method that evaluates a Device's CPU frequency
def evaluate_cpu_frequency(self, state):
    device_cpu_frequency = utils.get_attribute_from_object(
        self.devices[state], "cpu_frequency"
    )

    if device_cpu_frequency <= 0:
        return 0
    if device_cpu_frequency <= 5000:
        return 10
    if device_cpu_frequency <= 10000:
        return 20
    if device_cpu_frequency <= 15000:
        return 30
    if device_cpu_frequency <= 20000:
        return 40
    if device_cpu_frequency <= 25000:
        return 50
    if device_cpu_frequency <= 30000:
        return 60
    return 70

# Method that evaluates a Device's percentage parameters agains the Function Block's average consumption
def evaluate_percentage(self, device_resources, function_block_resources):
    if (
        self.compare_resource_parameters(
            device_resources, function_block_resources, 0
        )
        == 0
    ):
        return -50
    if (
        self.compare_resource_parameters(
            device_resources, function_block_resources, -90.0
        )
        == 0
    ):
        return -45
    if (
        self.compare_resource_parameters(
            device_resources, function_block_resources, -80.0
        )
        == 0
    ):
        return -40
    if (
        self.compare_resource_parameters(
            device_resources, function_block_resources, -70.0
        )
        == 0
    ):
        return -35
    if (
        self.compare_resource_parameters(
            device_resources, function_block_resources, -60.0
        )
        == 0
    ):
        return -30
```

Figure 8.19: Environment class reward associated methods

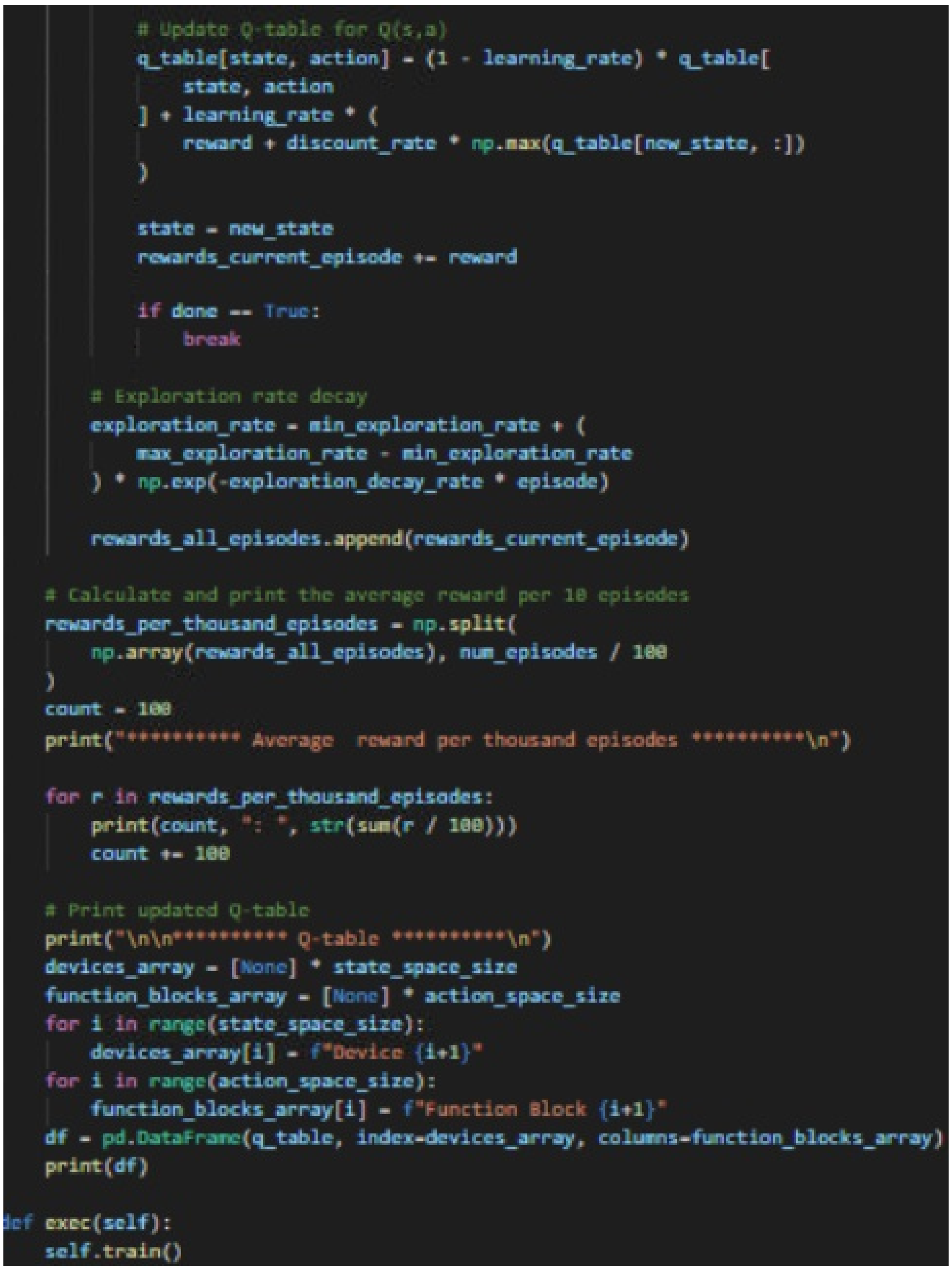


```
if (
    self.compare_resource_parameters(
        device_resources, function_block_resources, -50.0
    )
    == 0
):
    return -25
if (
    self.compare_resource_parameters(
        device_resources, function_block_resources, -40.0
    )
    == 0
):
    return -20
if (
    self.compare_resource_parameters(
        device_resources, function_block_resources, -30.0
    )
    == 0
):
    return -15
if (
    self.compare_resource_parameters(
        device_resources, function_block_resources, -20.0
    )
    == 0
):
    return -10
if (
    self.compare_resource_parameters(
        device_resources, function_block_resources, -10.0
    )
    == 0
):
    return -5
if (
    self.compare_resource_parameters(
        device_resources, function_block_resources, 0.00
    )
    == 0
):
    return 0
if (
    self.compare_resource_parameters(
        device_resources, function_block_resources, 10.0
    )
    == 1
):
    return 10
if (
    self.compare_resource_parameters(
```



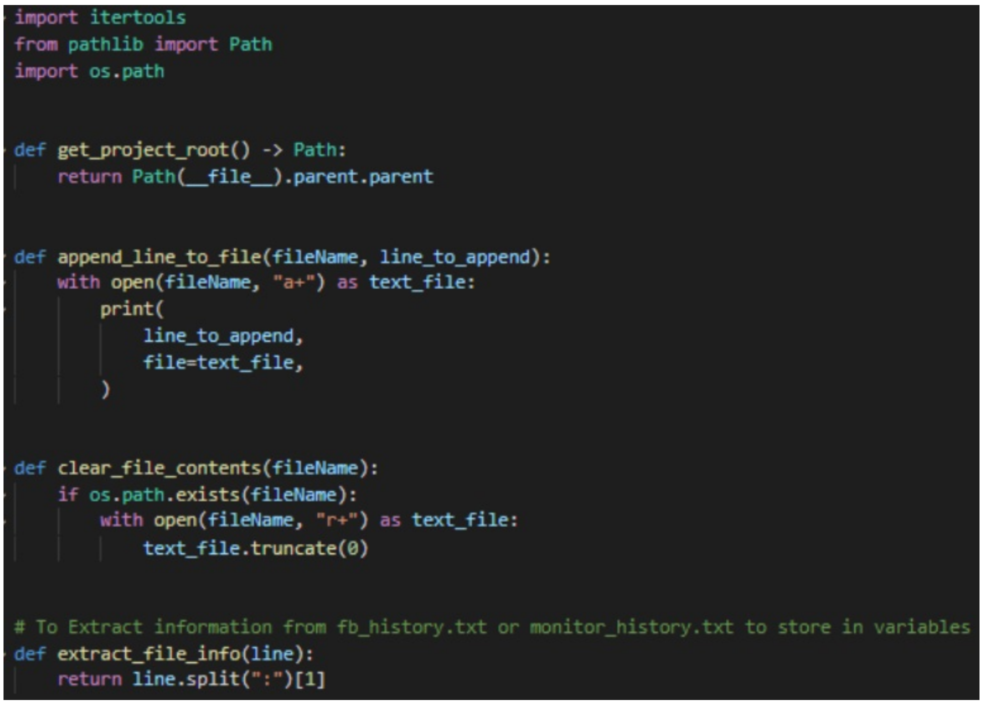
```
arning_algorithm.py > QLearningAlgorithm > exec
import environment
import numpy as np
import pandas as pd
import random
from stable_baselines.common.policies import MlpPolicy
from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines import PPO2


class QLearningAlgorithm:
    def __init__(self, devices: list, function_blocks: list):
        self.devices = devices
        self.function_blocks = function_blocks
        self.env = environment.BasicEnv(devices, function_blocks)

    def train(self):
        state_space_size = self.env.observation_space.n
        action_space_size = self.env.action_space.n

        q_table = np.zeros((state_space_size, action_space_size))

        print(q_table)

        num_episodes = 1000
        max_steps_per_episode = 10  # but it won't go higher than 1

        learning_rate = 0.1
        discount_rate = 0.99

        exploration_rate = 1
        max_exploration_rate = 1
        min_exploration_rate = 0.01

        exploration_decay_rate = 0.01  # if we decrease it, will learn slower

        rewards_all_episodes = []

        # Q-Learning algorithm
        for episode in range(num_episodes):
            state = self.env.reset()

            done = False
            rewards_current_episode = 0

            for step in range(max_steps_per_episode):

                # Exploration -exploitation trade-off
                exploration_rate_threshold = random.uniform(0, 1)
                if exploration_rate_threshold > exploration_rate:
                    action = np.argmax(q_table[state, :])
                else:
                    action = self.env.action_space.sample()

                new_state, reward, done, info = self.env.step(action)

                print("Info is: ", info)
```

Figure 8.21: Q-Learning algorithm initialization and beginning of algorithm



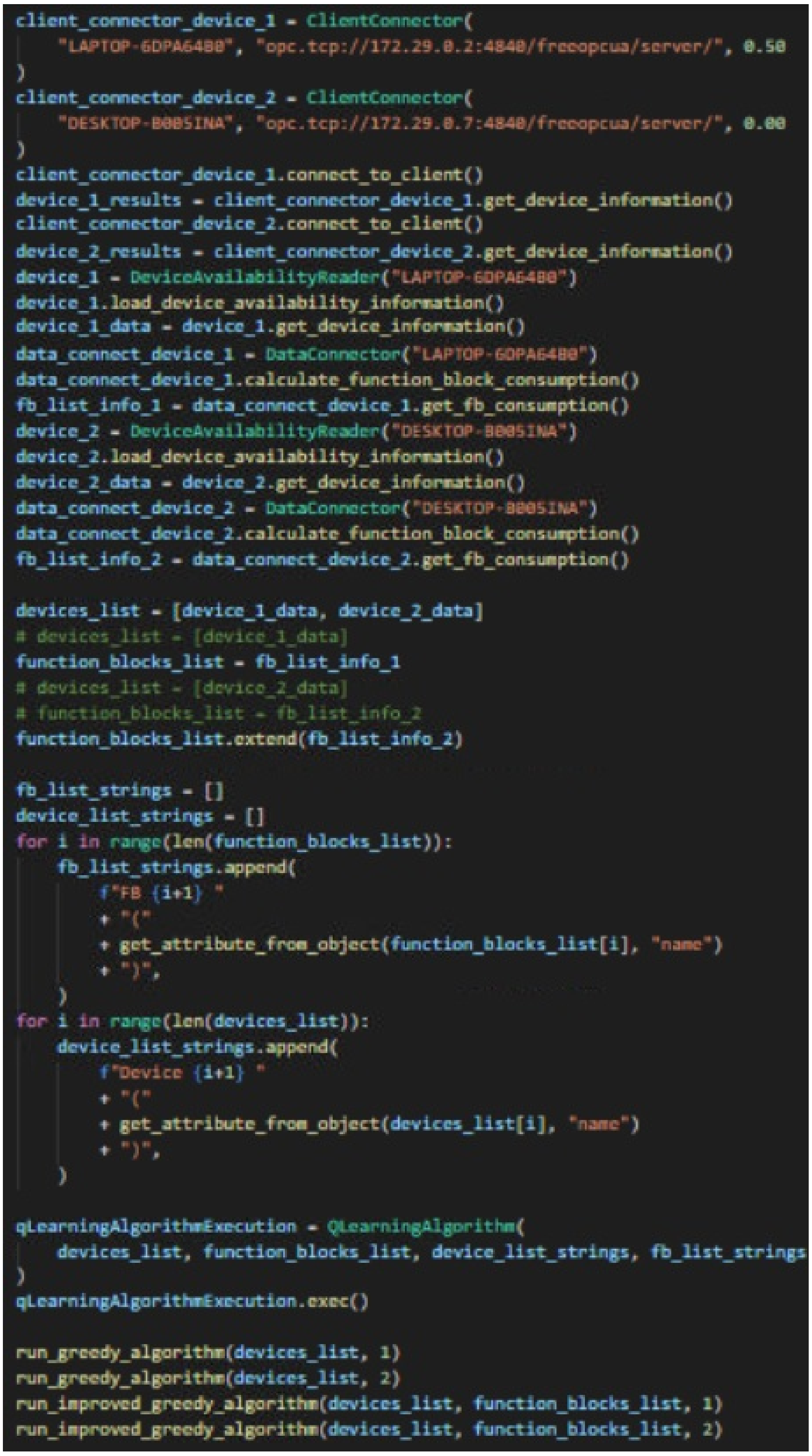
```
            # Update Q-table for Q(s,a)
            q_table[state, action] = (1 - learning_rate) * q_table[
                state, action
            ] + learning_rate * (
                reward + discount_rate * np.max(q_table[new_state, :])
            )

            state = new_state
            rewards_current_episode += reward

            if done == True:
                break

        # Exploration rate decay
        exploration_rate = min_exploration_rate + (
            max_exploration_rate - min_exploration_rate
        ) * np.exp(-exploration_decay_rate * episode)

        rewards_all_episodes.append(rewards_current_episode)

    # Calculate and print the average reward per 10 episodes
    rewards_per_thousand_episodes = np.split(
        np.array(rewards_all_episodes), num_episodes / 100
    )
    count = 100
    print("********** Average  reward per thousand episodes **********\n")

    for r in rewards_per_thousand_episodes:
        print(count, ": ", str(sum(r / 100)))
        count += 100

    # Print updated Q-table
    print("\n\n********** Q-table **********\n")
    devices_array = [None] * state_space_size
    function_blocks_array = [None] * action_space_size
    for i in range(state_space_size):
        devices_array[i] = f"Device {i+1}"
    for i in range(action_space_size):
        function_blocks_array[i] = f"Function Block {i+1}"
    df = pd.DataFrame(q_table, index=devices_array, columns=function_blocks_array)
    print(df)

def exec(self):
    self.train()
```

Figure 8.22: Q-Learning algorithm update of Q-table and showing results



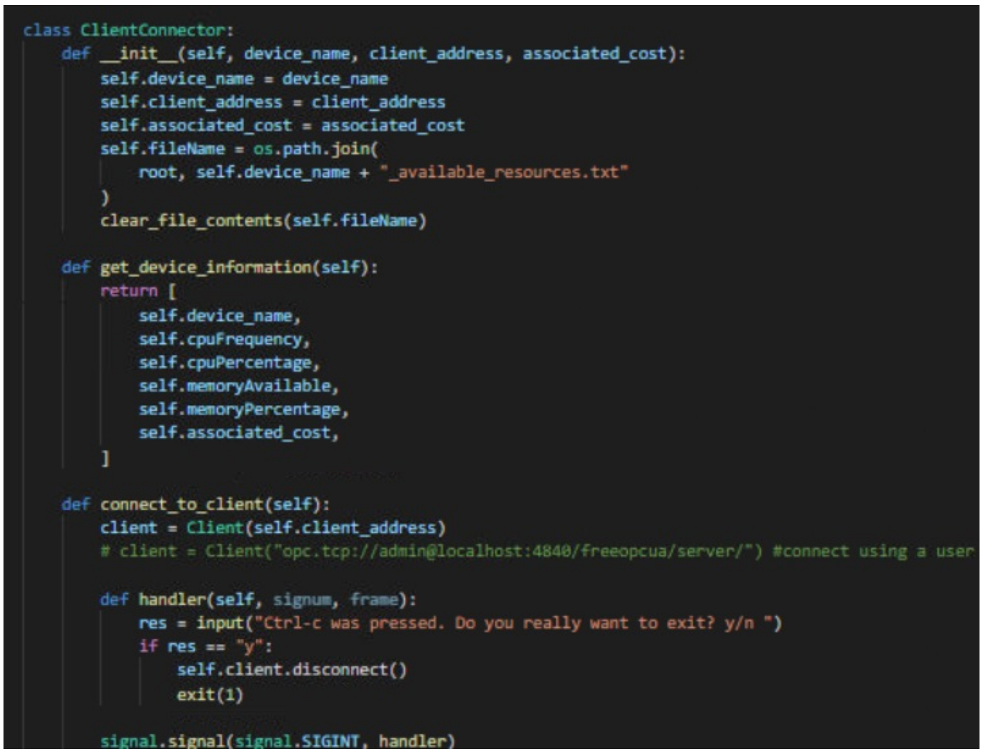
```python
import itertools
from pathlib import Path
import os.path


def get_project_root() -> Path:
    return Path(__file__).parent.parent


def append_line_to_file(fileName, line_to_append):
    with open(fileName, "a+") as text_file:
        print(
            line_to_append,
            file=text_file,
        )


def clear_file_contents(fileName):
    if os.path.exists(fileName):
        with open(fileName, "r+") as text_file:
            text_file.truncate(0)


# To Extract information from fb_history.txt or monitor_history.txt to store in variables
def extract_file_info(line):
    return line.split(":")[1]
```

Figure 8.23: Methods utilized in the mechanism to aid in file operations



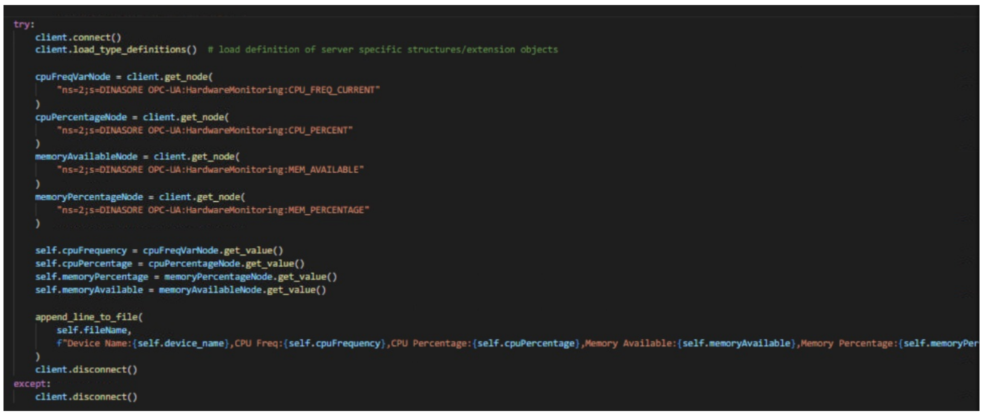
```
client_connector_device_1 = ClientConnector(
    "LAPTOP-6DPA6480", "opc.tcp://172.29.0.2:4840/freeopcua/server/", 0.50
)
client_connector_device_2 = ClientConnector(
    "DESKTOP-B005INA", "opc.tcp://172.29.0.7:4840/freeopcua/server/", 0.00
)
client_connector_device_1.connect_to_client()
device_1_results = client_connector_device_1.get_device_information()
client_connector_device_2.connect_to_client()
device_2_results = client_connector_device_2.get_device_information()
device_1 = DeviceAvailabilityReader("LAPTOP-6DPA6480")
device_1.load_device_availability_information()
device_1_data = device_1.get_device_information()
data_connect_device_1 = DataConnector("LAPTOP-6DPA6480")
data_connect_device_1.calculate_function_block_consumption()
fb_list_info_1 = data_connect_device_1.get_fb_consumption()
device_2 = DeviceAvailabilityReader("DESKTOP-B005INA")
device_2.load_device_availability_information()
device_2_data = device_2.get_device_information()
data_connect_device_2 = DataConnector("DESKTOP-B005INA")
data_connect_device_2.calculate_function_block_consumption()
fb_list_info_2 = data_connect_device_2.get_fb_consumption()

devices_list = [device_1_data, device_2_data]
# devices_list = [device_1_data]
function_blocks_list = fb_list_info_1
# devices_list = [device_2_data]
# function_blocks_list = fb_list_info_2
function_blocks_list.extend(fb_list_info_2)

fb_list_strings = []
device_list_strings = []
for i in range(len(function_blocks_list)):
    fb_list_strings.append(
        f"FB {i+1} "
        + "("
        + get_attribute_from_object(function_blocks_list[i], "name")
        + ")",
    )
for i in range(len(devices_list)):
    device_list_strings.append(
        f"Device {i+1} "
        + "("
        + get_attribute_from_object(devices_list[i], "name")
        + ")",
    )

qLearningAlgorithmExecution = QLearningAlgorithm(
    devices_list, function_blocks_list, device_list_strings, fb_list_strings
)
qLearningAlgorithmExecution.exec()

run_greedy_algorithm(devices_list, 1)
run_greedy_algorithm(devices_list, 2)
run_improved_greedy_algorithm(devices_list, function_blocks_list, 1)
run_improved_greedy_algorithm(devices_list, function_blocks_list, 2)
```

Figure 8.24: Main Class



```python
class ClientConnector:
    def __init__(self, device_name, client_address, associated_cost):
        self.device_name = device_name
        self.client_address = client_address
        self.associated_cost = associated_cost
        self.fileName = os.path.join(
            root, self.device_name + "_available_resources.txt"
        )
        clear_file_contents(self.fileName)

    def get_device_information(self):
        return [
            self.device_name,
            self.cpuFrequency,
            self.cpuPercentage,
            self.memoryAvailable,
            self.memoryPercentage,
            self.associated_cost,
        ]

    def connect_to_client(self):
        client = Client(self.client_address)
        # client = Client("opc.tcp://admin@localhost:4840/freeopcua/server/") #connect using a user

        def handler(self, signum, frame):
            res = input("Ctrl-c was pressed. Do you really want to exit? y/n ")
            if res == "y":
                self.client.disconnect()
                exit(1)

        signal.signal(signal.SIGINT, handler)
```

Figure 8.25: Client Connector Class



```python
try:
    client.connect()
    client.load_type_definitions()  # load definition of server specific structures/extension objects

    cpuFreqVarNode = client.get_node(
        "ns=2;s=DINASORE OPC-UA:HardwareMonitoring:CPU_FREQ_CURRENT"
    )
    cpuPercentageNode = client.get_node(
        "ns=2;s=DINASORE OPC-UA:HardwareMonitoring:CPU_PERCENT"
    )
    memoryAvailableNode = client.get_node(
        "ns=2;s=DINASORE OPC-UA:HardwareMonitoring:MEM_AVAILABLE"
    )
    memoryPercentageNode = client.get_node(
        "ns=2;s=DINASORE OPC-UA:HardwareMonitoring:MEM_PERCENTAGE"
    )

    self.cpuFrequency = cpuFreqVarNode.get_value()
    self.cpuPercentage = cpuPercentageNode.get_value()
    self.memoryPercentage = memoryPercentageNode.get_value()
    self.memoryAvailable = memoryAvailableNode.get_value()

    append_line_to_file(
        self.fileName,
        f"Device Name:{self.device_name},CPU Freq:{self.cpuFrequency},CPU Percentage:{self.cpuPercentage},Memory Available:{self.memoryAvailable},Memory Percentage:{self.memoryPer
    )
    client.disconnect()
except:
    client.disconnect()
```

Figure 8.26: Client Connector Class Thread